

博彦电脑交互式培训教程

内含全部源代码

博彦科技
BEYONDSOFT

内含
超值交互式
教学光盘

编程高手

Programming Master

Java

- ◆ 学会编程很简单
成为高手也不难
- ◆ 特别适合
初学 Java 编程者

北京大学出版社

Java 编程高手

北京博彦科技发展有限公司 著

北京大学出版社

内 容 简 介

本书系统介绍了使用 Java 进行应用程序开发以及 Java 小程序开发的基本方法和技巧。详细内容包括: Java 和 Visual J++ 简介, Visual J++ 开发环境, Java 编程基础, Java 类、接口和包的基础知识, Java 程序的分类, 编写 Java 小程序, 编写 Java 应用程序, 调试 Java 应用程序, 数据库编程, 开发 WFC 控件, 开发 COM 组件, 最后是一个 Visual J++ 应用程序开发实例。

本书内容循序渐进、实例丰富、解释细致, 不仅适合初学者和编程爱好者, 对于有编程经验的用户也会有一定的帮助。

版权所有, 翻印必究。

本书封面贴有北京大学出版社激光防伪标签, 无标签者不得销售。

书 名: Java 编程高手

图书著作者: 北京博彦科技发展有限公司

CD 著作者: 北京博彦科技发展有限公司

责任编辑: 杨锡林 张燕

出 版 者: 北京大学出版社

地 址: 北京市海淀区北京大学出版社 邮政编码 100871

发 行 者: 北京大学出版社

经 销: 各地新华书店、软件连锁店

CD 生产者: 上海联合光盘有限公司

图书印刷者: 中国科学院印刷厂

开本/规格: 787×1000 1/16 开本 印张: 34 字数: 734 千字

版次/印次: 2000 年 12 月第 1 版 2001 年 2 月第 2 次印刷

本 版 号: ISBN 7-900629-82-3/TP·61

定 价: 49.00 元(1CD 含配套书)

《编程高手》

——我的第一本程序设计教程

感谢您翻开我们编写的这套教程,请务必阅读下面的说明,以便确定《编程高手》系列丛书是否正是为您设计的。

作为专业从事计算机培训图书和教学软件的开发,博彦公司始终坚持以用户的学习需求为中心来开发产品。继我们推出《电脑学校》系列教材之后,很多读者反映,这种多媒体教学光盘和培训图书相结合的方式,学习效果非常好。同时,也有很多读者讲,在掌握了使用电脑及各种软件之后,他们迫切希望通过这种书盘结合的高效率学习方式学会编写程序。我们调查发现,读者对 Visual Basic、Visual C++、Java、ASP 和 Office VBA 的学习需求最为强烈。这几种语言也正是目前在各个领域中的应用最广泛的程序设计语言。应读者要求,我们开发了《编程高手》系列丛书。

这套《编程高手》系列丛书包含 5 本教程:

《Visual Basic 编程高手》

《Visual C++ 编程高手》

《Java 编程高手》

《ASP 编程高手》

《Office VBA 编程高手》

现在,让我们一起看看《编程高手》系列丛书的特点:

一、从低处着手、入门快。我们将这套教程定位于“我的第一本程序设计教程”。为此,我们专门针对初学编程的读者,在内容和形式上进行编排和规划。正因为这套教程是专门为初学编程的人员开发的,所以,即使您以前对编写程序一无所知,《编程高手》也能让您轻松上手。

二、从高处着眼、循序渐进。《编程高手》的起点低并不意味着书中只讲解浅显的基本编程知识,而是深入浅出、循序渐进地教您从按钮、菜单等基本元素的设计学起,到数据库访问、网络编程知识,甚至设计出一个网上商店。从而真正实现“学会编程很容易,成为高手也不难”。

三、结合大量实例,注重实践。计算机是一种工具,必须在实践中才能掌握,编写程序更是如此。《编程高手》提供了小到只包含一行代码的“Hello, World”程序,大到一个网上商店系统的大量实例源代码,并结合这些实例进行教学。另外,《编程高手》的每章最后,都有一节“更上一层楼”,启发您在本章实例的基础上,如何实现更强大、更深入的功能,从而达到举一反三的目的。

四、图书和多媒体教学光盘相结合。图书配套多媒体教学光盘是博彦图书最受读者欢迎的一大特色。很多读者可能都有过这样的感受:一边看书、一边在计算机上执行书中的步骤是很累,而且效率非常低的一件事,如果自己的计算机设置与书中有差异,或者书中的步骤不详细,稍有差错,就无法进行下去,一时间成就感全无,学习热情陡降。我们为您精心开发的多媒体教学光盘很好地解决了这个问题。《编程高手》光盘将编写每个程序的每步操作,每条语句,都直观、生动的展示给您,而且配有真人生动的讲解,让您看得清楚,听得明白,学得轻松。最值得一提的是,这套多媒体光盘继承了博彦独创的“试一试”教学方式,这种教学方式将读者带到真实的软件环境中,并像一个循循善诱的老师一样,指导您一步一步尝试操作,学练结合,在实践中牢固掌握所学知识。如果说编程序是电脑爱好者的终极乐趣,那么《编程高手》绝对是一个学识渊博的教师和不厌其烦的陪练,经过一番轻松、愉快的学习,您就可以基本掌握并能灵活运用。

经过数月的艰苦努力,我们终于要将这套《编程高手》奉献给我们的读者了。我们坚信这是一套可以极大提高您的学习效率的教程,但最终的评判还有待您的认可。如果您对我们的图书、光盘有什么意见和建议,欢迎来信告诉我们,正是由于广大读者批评和支持,我们才能不断改进和提高,为您提供更多、更好的学习教程。来信请寄:

100085 北京市海淀区上地6街17号
北京博彦科技发展有限公司客户服务部
电话: (010)86280136
传真: (010)86280141
电子邮件: support@beyondsoft.com.cn

前 言

Java 是一种面向对象的、多线程的、交互式的与平台无关的编程语言。它功能强大,表达能力强。Java 是从 C++ 发展而来的,它的语法比 C++ 更简单,减轻了编程人员的负担,使人用过之后耳目一新。随着互联网的迅猛发展,Java 以特有的小程序风靡全球,成为 Internet 中标准的程序设计语言。

Visual J++ 是 Microsoft 为 Java 语言所开发的用于 Windows 编程的集成开发环境。Visual J++ 语言是帮助我们方便快捷地开发基于 Web 的小程序和 Windows 环境下的应用程序的有力工具。

本书主要介绍关于 Java 和 Visual J++ 的知识。其中第 1 章简要介绍 Java 和 Visual J++, 第 2 章介绍 Visual J++ 开发环境,第 3 章介绍 Java 编程基础,第 4 章介绍 Java 的类、接口和包,第 5 章介绍 Java 应用程序分类,第 6 章和第 7 章介绍如何编写 Java 小程序,第 8 章和第 9 章介绍如何编写 Java 应用程序,第 10 章介绍如何调试 Java 应用程序,第 11 章介绍数据库编程,第 12 章介绍如何开发 WFC 控件,第 13 章介绍如何开发 COM 组件,第 14 章是一个 Visual J++ 应用程序开发的实例。

本书配套光盘的 Sample 文件夹中,包括本书各章所创建的示例程序的源代码,可供读者参考。

北京博彦科技发展有限公司

2000 年 11 月

编程高手
Programming Master

博彦科技
BEYONDSOFT

Java

学会编程很容易，成为高手也不难！

本书系统介绍了使用 Java 进行应用程序开发以及小程序开发的基本方法和技巧。详细内容包括：Java 和 Visual J++ 简介，Visual J++ 开发环境，Java 编程基础，Java 类、接口和包的基础知识，Java 程序分类，编写 Java 小程序，编写 Java 应用程序，调试 Java 应用程序，数据库编程，开发 WFC 控件，开发 COM 组件，最后是一个应用程序开发实例。

内含光盘使用方法：

- 了解环境，快速搭建开发环境

要使用内含光盘，您需要：

- 具有 Pentium 166 以上或同等处理速度的计算机
- 16MB 以上内存
- 20MB 以上硬盘空间
- Windows 95/NT/2000
- 网络浏览器：Netscape 4.0 或 Internet Explorer 3.0 以上
- 光盘驱动器：40X 或 80X 均可



通过这个按钮，可以在真实的环境中实践本书讲述的内容。注意，要使用此功能时必须安装相应的软件。

通过这个按钮可以打开本书光盘的目录，浏览并选择其他的学习内容。

这是一个帮助按钮，通过这个按钮可以在程序自动演示和平地练习操作两种学习方式中切换。



目 录

第1章 Java与Visual J++简介	1
1.1 Internet与Java	2
1.1.1 Internet发展的障碍	2
1.1.2 为什么选择Java	3
1.1.3 Java产生的历史与现状	4
1.2 Java语言的特点	6
1.2.1 简单	6
1.2.2 面向对象	7
1.2.3 分布式	7
1.2.4 健壮	8
1.2.5 与平台无关	8
1.2.6 安全	8
1.2.7 可移植	9
1.2.8 解释型	10
1.2.9 高性能	10
1.2.10 多线程	11
1.2.11 Java的动态特性	11
1.3 Java与C/C++的比较	11
1.4 为什么用Visual J++进行Java编程	13
第2章 快速熟悉Visual J++	17
2.1 安装Visual J++	18
2.1.1 对硬件的要求	18
2.2.2 运行安装程序	18
2.2 集成开发环境的优点	25
2.2.1 解决方案和工程文件系统	26
2.2.2 Windows基类WFC和J/Direct	26
2.2.3 Internet支持	27
2.2.4 COM支持	27
2.2.5 向导和生成器	27
2.2.6 数据访问	28

2.2.7 打包和部署	28
2.3 熟悉 Visual J++ 的用户界面	28
2.3.1 进入 Visual J++ 的用户界面	28
2.3.2 熟悉 Visual J++ 的用户界面	30
2.4 从 Hello world 开始	36
2.4.1 创建工程和窗体	36
2.4.2 添加显示的文本	37
2.4.3 运行程序	40
更上一层楼	41
第3章 Java 编程概况	43
3.1 程序结构	44
3.1.1 命令行的“Hello World”Java 程序	44
3.1.2 注释	46
3.2 标识符	46
3.3 保留字	47
3.4 数据类型	48
3.4.1 常量	48
3.4.2 变量	48
3.4.3 整型(integral type)	50
3.4.4 浮点型(float-pointtypes)	52
3.4.5 布尔变量(Boolean)	52
3.4.6 字符型(char type)	53
3.4.7 字符串	54
3.4.8 数组	57
3.5 运算符和表达式	58
3.5.1 算术运算符	59
3.5.2 关系运算符	60
3.5.3 布尔逻辑运算符	60
3.5.4 位运算符	61
3.5.5 表达式	61
3.5.6 运算符的优先级	62
3.6 Java 流控制	63
3.6.1 分支语句	64
3.6.2 循环语句	65

3.6.3 标号和转移语句	67
更上一层楼	69
第4章 Java 类、接口、包	71
4.1 Java 类与对象	72
4.1.1 面向对象编程的基本概念	72
4.1.2 Java 类	74
4.2 接口	85
4.2.1 为什么使用接口	85
4.2.2 接口的定义	87
4.2.3 接口的实现	89
4.3 包	91
4.3.1 为什么要使用包	91
4.3.2 包与类名	92
4.3.3 包与目录	93
4.3.4 包(package)语句	94
4.3.5 import 语句	95
更上一层楼	96
第5章 Java 应用程序分类	97
5.1 小程序和应用程序比较	98
5.1.1 Java 应用程序分类	98
5.1.2 小程序和应用程序的比较	98
5.2 小程序	100
5.2.1 Applet 类的继承关系	100
5.2.2 创建 Java 小程序	100
5.2.3 运行结果	106
5.3 应用程序	107
5.3.1 创建应用程序	107
5.3.2 运行结果	109
更上一层楼	109
第6章 Java 小程序编程入门	111
6.1 一个简单的例子	112
6.1.1 新建工程	112
6.1.2 给工程添加类	112
6.1.3 给类中引入包	114

6.1.4	给类中添加方法	114
6.1.5	在方法中添加自己的语句	117
6.1.6	运行结果	118
6.2	Applet 类	119
6.2.1	理解程序	119
6.2.2	java.applet 包与 Applet 类	119
6.2.3	Applet 类中的方法	120
6.3	java.awt 包中的类	122
6.3.1	理解程序	122
6.3.2	java.awt 包中的类	123
6.4	事件处理	125
6.4.1	处理事件	125
6.4.2	处理常用事件	127
6.4.3	在小程序中添加事件处理代码	129
6.5	把小程序嵌入到 Web 页	131
6.5.1	向工程中加入 Web 网页	132
6.5.2	基本 HTML 标记	136
6.5.3	<APPLET> 标记	138
6.6	使用 Applet 参数	140
6.6.1	<PARAM> 标记	141
6.6.2	在小程序中使用参数	141
6.7	精彩实例	144
6.7.1	滚动字幕	145
6.7.2	水波倒影	149
6.7.3	小丸子时钟	152
	更上一层楼	157
第 7 章	Java 小程序编程进阶	159
7.1	小程序的界面组件	160
7.1.1	按钮(Button)	162
7.1.2	文本框(TextField)和文本域 TextArea)(TextArea)	169
7.1.3	复选框(Checkbox)和复选框组(CheckboxGroup)	173
7.2	布局管理	178
7.2.1	布局管理器	178
7.2.2	FlowLayout(流布局管理器)	179

7.2.3 BorderLayout(边框布局管理器)	180
7.2.4 GridLayout(网格布局管理器)	181
7.2.5 GridBagLayout(网袋布局管理器)	183
7.2.6 综合使用	185
7.3 多线程小程序	187
7.3.1 什么是多线程	187
7.3.2 在小程序中创建线程	188
7.3.3 线程的方法	190
7.3.4 同步	192
7.3.5 多线程在小程序中应用的例子	197
更上一层楼	199
第8章 Java 应用程序编程入门	201
8.1 创建和显示窗体	202
8.1.1 窗体简介	202
8.1.2 创建和显示窗体实例	203
8.1.3 创建工程和应用程序的主窗体	204
8.1.4 创建启动屏幕窗体	205
8.1.5 将代码添加到启动屏幕的窗体中	206
8.1.6 添加启动屏幕窗体的方法和事件处理程序	210
8.1.7 编写代码以显示启动屏幕	215
8.2 创建屏幕保护程序	216
8.2.1 创建工程及其主窗体	217
8.2.2 将成员变量添加到屏幕保护程序窗体中	218
8.2.3 将功能添加到屏幕保护程序中(一)	219
8.2.4 将功能添加到屏幕保护程序中(二)	221
8.2.5 编译、打包及发布屏幕保护程序	225
8.3 菜单操作	226
8.3.1 创建菜单	227
8.3.2 创建菜单事件程序	235
8.3.3 创建环境菜单	247
8.4 创建工具栏	255
8.4.1 在新工程中打开 WFC Jot 应用程序	256
8.4.2 添加 imageList 控件及其图象	256
8.4.3 添加 ToolBar 控件及其按钮	257

8.4.4 处理工具栏的事件.....	259
8.5 创建状态栏	262
8.5.1 在新工程中打开 WFC Jot 应用程序	263
8.5.2 添加 StatusBar 控件并创建其窗格	263
8.5.3 添加支持状态栏的代码	264
更上一层楼	265
第 9 章 Java 应用程序编程进阶	267
9.1 使用控件	268
9.1.1 什么是控件	268
9.1.2 综合使用 Windows 基本控件	269
9.2 在应用程序中加入帮助	289
9.2.1 加入帮助前的准备	290
9.2.2 打开 HockeyPlayerScout 应用程序并添加帮助控件	290
9.2.3 添加 Help 按钮的事件处理程序及代码	291
9.2.4 添加支持 F1 键和“What’s This”帮助的代码	292
9.2.5 运行结果	294
9.3 在应用程序中支持拖放	296
9.3.1 创建工程及其窗体	296
9.3.2 创建窗体控件的事件处理程序	298
9.3.3 添加拖放支持代码	298
9.3.4 拖放例子运行结果	301
9.3.5 拖放操作的改进	301
9.4 在应用程序中使用 ActiveX 控件	302
9.4.1 ActiveX 技术背景	303
9.4.2 NumText 控件说明	304
9.4.3 创建工程并设计窗体	305
9.4.4 添加 ActiveX 控件	306
9.4.5 添加事件处理程序及代码	307
9.4.6 编译运行	309
9.5 Java 应用程序与 Java 小程序的混合	309
9.5.1 应用程序与小程序的混合编程	309
9.5.2 Java 应用程序与 Java 小程序例子	310
9.5.3 HelloJava.exe	314
更上一层楼	316

第 10 章 调试	317
10.1 调试环境介绍	318
10.1.1 调试的工具和窗口	318
10.1.2 调试前的准备	320
10.2 调试代码	321
10.2.1 断点	321
10.2.2 执行到光标处	324
10.2.3 在源代码中单步运行	324
10.2.4 Watch 窗口	326
10.2.5 Immediate 窗口	327
10.3 调试器的其他窗口	327
10.3.1 调试例子	328
10.3.2 Output 窗口	328
10.3.3 Autos 窗口	331
10.3.4 Locals 窗口	332
10.3.5 Call Stack 窗口	333
10.4 Java 小程序的调试技巧	333
10.4.1 ex06d 例子再调试	334
10.4.2 设置 Java 小程序所在工程的属性	334
10.4.3 跟踪 Java 小程序的参数	335
10.5 调试过程	336
10.5.1 编译、运行	336
10.5.2 调试、修改	337
10.6 调试异常处理	337
10.6.1 Java Exceptions 对话框	337
10.6.2 异常设置(Exception Settings)	338
更上一层楼	339
第 11 章 数据库编程	341
11.1 为什么用 ADO	342
11.1.1 早期的数据库	342
11.1.2 为什么选择 ADO	342
11.2 数据控件与数据库绑定	343
11.2.1 创建工程并添加窗体	343
11.2.2 将控件添加到窗体中	344

11.2.3	将 DataSource 控件与数据库联系起来	348
11.2.4	将控件绑定到数据库中	350
11.2.5	为按钮添加事件处理程序及代码	351
11.3	使用数据控件	355
11.3.1	新建工程	356
11.3.2	更改 TreeView 控件并添加 DataSource 和 DataGrid 控件	357
11.3.3	将代码添加到实例中	357
11.4	直接访问数据库	361
11.4.1	打开 HockeyScout 工程并准备代码	362
11.4.2	添加 DataSource 对话框	363
11.4.3	创建 PlayerSelect 对话框	363
11.4.4	添加访问数据库的代码	365
11.5	使用 Data Form Wizard	376
11.5.1	创建工程并显示 Data Form Wizard	376
11.5.2	指定数据库类型及名称	376
11.5.3	指定窗体类型	378
11.5.4	选择主要的和详细的记录源及字段	379
11.5.5	选择记录源关系及控件	381
11.5.6	完成向导,检查窗体并运行实例	383
	更上一层楼	384
第 12 章	WFC 控件开发	387
12.1	子类化控件	388
12.1.1	创建控件工程	389
12.1.2	创建 ClassInfo 类及 Value 属性	390
12.1.3	定义 NonNumberEntered 事件	392
12.1.4	将代码添加到 GetValue 方法中	393
12.1.5	使用 Class Outline 忽略方法	394
12.1.6	将控件添加到窗体中	397
12.2	自定义控件	398
12.2.1	创建控件工程	399
12.2.2	将代码添加到构造函数中	399
12.2.3	添加属性和事件	400
12.2.4	忽略 Control 类的方法	402

12.2.5 将 ClockTestForm 窗体添加到工程中	405
12.3 组合控件	406
12.3.1 使用控件模板创建控件工程	407
12.3.2 将控件添加到 UserControl 中	408
12.3.3 创建控件属性	410
12.3.4 创建 DefaultState 属性对话框	415
12.3.5 将 DefaultStateEditor 值编辑器添加到工程中	417
12.3.6 将公共方法添加到 AddressProvider 类中	419
12.3.7 添加测试控件的窗体	420
12.4 WFC 到 ActiveX 的转换	421
12.4.1 打开控件工程	422
12.4.2 将控件类注册为 COM 类	422
12.4.3 将控件打包到 COM DLL 中	423
12.4.4 注册 COM DLL	423
12.4.5 创建包含控件的 Visual Basic 工程	423
更上一层楼	425
第 13 章 COM 组件开发	427
13.1 关于 COM 的相关概念	428
13.2 COM 服务器	429
13.2.1 使用 COM 模板创建工程	429
13.2.2 在类中添加方法	430
13.3 WFC COM 客户应用程序	433
13.3.1 创建 WFC 应用程序工程	434
13.3.2 设计窗体的用户界面	434
13.3.3 导入 COM 组件	440
13.3.4 添加支持代码	441
13.4 用户接口组件	443
13.4.1 创建工程	444
13.4.2 将对话框添加到工程中	444
13.4.3 将方法添加到 COM 类中	444
13.4.4 编译 COM 客户应用程序	446
13.4.5 导入 FormReuse COM 组件	446
13.5 数据库 COM 组件	448
13.5.1 创建 COM 工程	449

13.5.2	设计 CustomerSearchDlg	449
13.5.3	将支持代码添加到 CustomerSearchDlg 中	453
13.5.4	将代码添加到 CustomerSearch COM 类中	461
13.5.5	编译 CustomerSearchEngine 客户应用程序	462
13.6	第三方 COM 组件	466
13.6.1	创建 WFC 应用程序工程	467
13.6.2	导入 Microsoft Word COM 组件	467
13.6.3	将支持代码添加到应用程序中	467
	更上一层楼	471
第 14 章	Visual J++ 应用程序的高级实例	473
14.1	实例背景	474
14.2	前期规划	475
14.3	开发计划	477
14.4	开发过程	478
14.4.1	准备	478
14.4.2	Order.java 对话框用户界面设计	479
14.4.3	COM 组件设计	482
14.4.4	代码的重要部分: Order.java 窗体代码	482
14.4.5	主要代码: MDIMain 窗体代码	502
14.4.6	应用程序的登录对话框	512
14.4.7	Previous Orders 窗口	515
14.4.8	Order Details 对话框	517
14.4.9	Password Change 和 About 对话框	518
14.5	程序运行	519
	更上一层楼	521

第 1 章 Java 与 Visual J++ 简介

知识要点:

- ◆ Internet 与 Java 的关系
- ◆ Java 语言的特点
- ◆ Java 与 C/C++ 的不同之处
- ◆ 为什么用 Visual J++ 进行 Java 编程

在本章中,我们将从 Internet 与 Java 的关系开始,介绍 Java 的产生背景、历史与现状,以及在 Internet 时代为什么选择 Java。然后对 Java 语言的特点进行介绍,对 Java 与 C/C++ 进行比较,最后列举用 Visual J++ 进行 Java 编程的优势。



光盘 参阅本书配套光盘的【基础知识】部分可交互学习与本章相关的知识。

1.1 Internet 与 Java

Java 的发展是与 Internet 密不可分的,是 Internet 促成了 Java? 还是 Java 使 Internet 更加精彩? 现在看来这两方面兼而有之。这一节我们就从 Internet 开始讲起,使大家对 Java 产生的背景有个基本的了解。

1.1.1 Internet 发展的障碍

90 年代以来,随着计算机技术特别是计算机网络技术的迅猛发展,单机时代已经过去,取而代之的是互联网时代。全世界各个国家和地区从未像今天这样被紧密联系在一起,人与人可以跨越空间尽情交流,Internet 已经深入到世界的每个角落,Internet 用户数目以惊人的速度迅速增长。随着技术的不断进步,Internet 应用领域的不断拓宽,众多的 Internet 服务层出不穷,电子邮件、WWW 服务、文件传输服务(FTP)、远程登录服务(Telnet)、网络新闻、电子商务、点播电视等,正逐渐为越来越多的人所使用。

尽管 Internet 在 60 年代就出现了,但它巨大的潜力在 90 年代才被认识到,WWW(World Wide Web)技术在这里起到了举足轻重的作用。早期的 Internet 只有简单的命令行界面,使用起来很不方便,通常只是一些专家和科研人员使用,一直都没有得到普及。WWW 的出现改变了这一局面,它将 Internet 资源当作链接文档,从而改变了人们访问信息的方式,人们从此可以足不出户地主动去寻找所需的信息,在不必学习复杂命令的情况下得到 Internet 服务。由于 WWW 服务方便易用,所以从网上获取信息变得非常快捷。随着 WWW 服务的迅速普及,Internet 进入了普通百姓的家庭。不过,许多人接触 Internet 都是通过 Web 浏览器(如 Internet Explorer 和 Netscape Navigator),因而常常将 Internet 和 WWW 混淆起来。实际上,WWW 比 Internet 的出现晚得多,直到 1992 年才由欧洲高能粒子物理实验室的 Tim Berners Lee 建立起来。最早的 WWW 浏览器则是美国国家超级计算应用中心开发的 Mosaic 浏览器,它像野火一样迅速红遍了整个 Internet,以后又出现了更好的 WWW 浏览器,使上网变得更加简单方便。

Internet 使我们进入一个新时代,但 Internet 并不是完美无缺的。Internet 虽然带给我们种种便利,但它还是不能充分满足我们日益增长的需要。Internet 在内容提供方面的不足之处主要体现在 Internet 上的内容是静止和被动的,而且内容的设计必须考虑到用户计算机和浏览器的情况。在 Internet 上大多数网页是静止的,就像是高速公路上公告牌一样,每个人去看,都是看到同样的内容。即

便是通过 CGI(公共网关接口)、ASP(活动服务器文档)等技术能够根据用户的请求动态产生相应的页面,传递给用户的页面其实还是静止的,实际上并没有交互性,客户端的请求还要送回服务器。这样的页面不能像电视那样随时都在变化,不能让用户通过网络随时了解股市的行情。用户与 Web 页面不能进行交互操作,而只能查看所提供的页面。在 Java 之前,设计 Web 页面时必须要考虑用户计算机和浏览器的情况,例如有的 Web 页面包含图形、声音、视频等内容,如果访问该 Web 页面的用户的浏览器不是最新版本,不支持这些最新的功能,那么他就有可能看不了这个页面,这也是为什么许多 Web 页面上都要注明“请使用 Internet Explorer 4.0 以上版本的浏览器”、“如看不到上面的效果显示,请在这里下载相应的插件”等字样。如果不解决这些问题,那么它们势必成为 Internet 发展的障碍。

幸运的是,Java 的出现使这一切有了改观,Java 给 WWW 带来了一场革命。它使 Web 页更加丰富多姿,Web 页面再也不是像以前那样呆板了,Java 给 Web 页面带来了动态和交互,Java 还使网上应用程序的编写变得非常容易。然而,Java 的意义远不止于此,Java 凭借它众多引人注目的特点,逐渐成为 Internet 上最受欢迎的开发与编程语言。一些著名的计算机公司纷纷购买了 Java 语言的使用权,如 MicroSoft、IBM、Netscape、Novell、Apple、DEC、SGI 等,Java 语言还被美国的著名杂志《PC Magazine》评为 1995 年十大优秀科技产品。那么,在 Internet 时代为什么大家都一致选择了 Java 呢?

1.1.2 为什么选择 Java

首先,选择 Java 是当今 Internet 发展的客观要求。前面说过,Java 在解决 Internet 上内容静止、被动和内容依赖浏览器问题方面有卓越表现。例如在实现动态网页方面,Java 可以轻松地在 Internet 上实现股票图、表的实时显示。Java 还使网页设计者无需考虑用户计算机和浏览器的具体配置,而现在的浏览器都是支持 Java 的。这样,使用 Java 的网页就可以正确地运行在任何硬件平台和操作系统上。这两方面正是反映了 Internet 发展的客观要求。

其次,Java 是一门各方面性能都很好的编程语言,可以制作大部分网络应用程序系统,而且与当今流行的 WWW 浏览器结合得很好。它的基本特点是简单、面向对象、分布式、解释型、健壮、安全、与平台无关、可移植、高性能、多线程和动态,十分适合在 Internet 环境上开发应用系统。单就“与平台无关”这一特点来讲,由于 Java 是一种与平台无关的语言,因此用 Java 开发的网络应用系统可以在各种平台上运行,这就大大提高了开发效率,减少了重复劳动。Java 程序的平台无关性使程序员们从可移植性的苦恼中摆脱出来,只要编写一个 Java 程序,

可以在任何平台上运行,无论是 Windows 环境,还是 Unix 环境。因此有人形象地比喻,“Java 是 Internet 上的世界语”。

第三,Java 语言将对未来软件的开发产生重大影响。传统的软件往往都是与具体的实现环境有关,换了一个环境就需要做一番改动,耗时费力,而 Java 语言能使以前开发的软件运行在不同的机器上,只要所用的机器能提供 Java 语言解释器即可。因此,Java 语言正在引起一场软件革命,具体体现在 Java 的软件需求分析可将用户的需求进行动态的、可视化描述。用户的需求将不受地区、行业、部门、爱好的影响,都可以用 Java 语言描述清楚,这样能够给设计者提供更加直观的要求。Java 的软件完全可以采用面向对象的方法来开发,符合软件开发的工业标准。Java 语言的动画效果远比 GUI 技术效果逼真,而且可以充分共享 Internet 上的动画资源。用 Java 语言开发的软件最终产品可以具有可视化、可听化、可操作化的效果,这要比电视的效果更为理想。因为它可以做到即时、交互、动画与动作,这是电视难以做到的。另外,用 Java 语言开发的软件,其效益和开发价值都有比较明显的优势。

第四,现在开发并推广 Java 技术的 Sun 公司,通过颁发许可证的办法允许各家公司把 Java 虚拟机和 Java 的小程序类库嵌入他们开发的操作系统。这样,开发人员就能选择多种平台来使用 Java 语言编程,不同的用户可以脱离 WWW 浏览器来运行 Java 应用程序。这无疑将大力促进 Java 的发展,同时也为 Java 语言的应用开拓了极为广阔的前景。

1.1.3 Java 产生的历史与现状

Java 的历史

Java 起源于 Sun 公司的一个叫 Green 的项目,最初的目的是为家用消费电子产品开发一个分布式代码系统,使其成为智能的电子产品。这样,人们可以通过 Internet 把 E-mail 发给电冰箱、电视机等家用电器,对它们进行控制,并和它们进行交互式操作。

1991 年, Sun 公司的杰出工程师 Jame Gosling、Bill Joe 等人发现消费类电子产品的用户不同于工作站的用户,他们不关心 CPU 的型号,而只需要一个建立在标准基础之上的与硬件平台无关的可选方案。为了做到平台无关性, Gosling 首先想到改写 C++ 的编译器,但不久他就发现 C++ 是无法满足需要的。因为 C++ 太复杂,而且在 C++ 中有着巨大的安全隐患,例如指针的使用,没有数组越界的检查,缺乏自动的内存管理,过于灵活的数据类型转换等。这使他开始考虑开发一种新的面向对象语言,于是 Oak 项目组就基于 C++ 开发出一种新的用于网络

的精巧而安全的语言——Oak(Java的前身)。Oak最初是一种用来给蜂窝电话及遥控器之类的设备编程的语言。Oak程序可以在需要时下载,这样就可以取代以往设备出厂前预先的编程工作,而且当设备新增功能时,客户可以立即用上,而不用再将设备送回工厂。1993年,Sun采用此技术建立了遥控器样机,当时尽管令人鼓舞,但是没有得到销售商的广泛支持,这个在技术上很成功的产品在市场上却遇到了挫折。

1994年,Internet和WWW在全球如火如荼地发展起来,Oak项目组开始认识到它的可下载技术可以用到Web上,于是决定将他们的技术用于WWW服务。他们把这种语言的名称改为Java,Java是咖啡的俚语,而这种语言也的确像咖啡一样可口、令人耳目一新。Oak项目组决定在已有工作的基础上研制一种新的浏览器。

在当时WWW服务还是静态的,缺少交互性,只是一些静态的图像和文本。在Web页上也的确出现了一些诸如简单的绘画程序的CGI脚本,但实际上并没有交互性。客户端的请求还要送回服务器,这样就给服务器增加了额外的负担。如果程序能够下载并能在客户端的浏览器上运行,那么服务器的负担就会减轻,从而能够提供更多的文档服务。这种浏览器就是Oak项目组当时想要建立的浏览器,后来他们用Java编制了HotJava浏览器。1995年5月,HotJava浏览器发表后,引起了巨大的轰动。HotJava是第一个具有自动装载和运行Java程序的浏览器。HotJava在WWW上的出色表现使人们对Java产生了很大兴趣,许多公司,如Netscape、Microsoft、IBM、Oracle等,纷纷从Sun那里取得Java的许可证,以便把Java技术应用到他们的产品中。一些当年对Green项目有兴趣的消费电子公司也纷纷联系Java的使用许可证。Java技术终于得到了它应有的地位,从此Java开始全面进军Internet。

Java的现状

由于Java是新一代的面向对象的程序设计语言,它的平台无关性特别适合于Internet应用程序开发。因此,连接Internet,用Java编程,很快成为技术人员的一种时尚。现在,Java作为软件开发的一种革命性的技术,其地位已被确立,这表现在以下几个方面:

- ◆ 计算机产业的许多大公司购买了Java的许可证,包括IBM、Apple、DEC、Adobe、SiliconGraphics、HP、Oracle、Toshiba以及Microsoft。这一点说明,Java已得到了工业界的认可。
- ◆ 众多的软件开发商开始支持Java的软件产品。例如Borland公司开发的基于Java的快速应用程序开发环境J++ Builder。Borland公司的这一举措推动了Java进入PC机软件市场。Sun公司的Java开发环境——Java Workshop

已经发布,数据库厂商 Oracle、Sybase、Illustra、Versant 等也都开发出 CGI 接口以支持 HTML 和 Java。现在是以网络为中心的时代,如果不支持 HTML 和 Java,那么应用程序的应用范围就会受限制(只能用于相同的硬件平台和操作系统)。

- ◆ Intranet 正在成为企业信息系统最佳的解决方案,而其中 Java 将发挥不可替代的作用。Intranet 的目的是把 Internet 技术应用于企业内部的信息及管理系统。无论用户使用何种类型的机器和操作系统,界面都是统一的 WWW 浏览器,而数据库、Web 页面、应用程序则存在 WWW 服务器上,无论是开发人员,还是管理人员或用户,都可以受益于该解决方案。

Java 较通用的编译环境有 JDK(Java Develop Kit)与 JWS(Java Workshop),还有很多公司正在开发 Java 语言的编译器与集成环境。预计不久,Java 语言的效率将会进一步提高,用户用 Java 编程将变得更方便。

Java 语言有着广泛的应用前景。例如,面向对象的事件描述、处理及综合,计算过程的可视化,动态画面的设计,图形图像的调用,交互操作的设计,Internet 的系统管理功能模块的设计,Web 页面的动态设计和交互操作设计,Intranet 上的软件开发,与各类数据库连接查询的 SQL 语句实现等。

1.2 Java 语言的特点

Java 到底是一种什么样的语言呢?Java 是一种简单的、面向对象的、分布式的、健壮的、与平台无关的、安全的、可移植的、解释型的、高性能的、多线程的、动态的语言。这是 Sun 公司的 Java 开发者们在 Java 语言的白皮书中给 Java 所下的定义,他们把 90 年代关于程序设计语言几乎所有重要的词语都放到这一定义中,下面我们来看究竟 Java 是否不负重望地具备了所有这些令人振奋的特点。

1.2.1 简单

Java 是一种简单的语言。它用到的概念不多,而且多为程序员所熟悉。如果您是一名程序员,掌握 Java 对您来说是易如反掌的事。即使您没有学过任何编程语言,学习 Java 也要比学习 C++ 要容易得多。

Java 语言是一种面向对象的语言,它通过提供最基本的方法来完成指定的任务,只需理解一些基本的概念,就可以用它编写出适合于各种情况的应用程

序。Java 略去了运算符重载、多重继承等模糊的概念,适合于在小型机上运行。它的基本解释器及类的支持只有 40KB 左右,加上标准类库和线程的支持也只有 215KB 左右。

Java 语言简单明了,这得益于它最初是为对家用电器进行集成控制而设计的。为了保证这种简单性,Java 摒弃了 C++ 中许多复杂的、冗余的、有二义性的概念,例如指针、操作符重载、多继承、数据类型自动转换等。为了将程序员从复杂的内存管理的负担中解脱出来,同时也是为了减少错误,Java 使用了自动内存垃圾收集机制,即 Java 自己来收集、释放内存中的无用的块。另外,Java 还提供了丰富的类库。这些都大大简化了程序员的设计和管理工作。

1.2.2 面向对象

面向对象可以说是 Java 最重要的特性。与 C++ 相比,Java 有着更强的面向对象特性,是一种纯粹的面向对象语言。一般我们使用的一些所谓的面向对象的编程语言,如 C++、Pascal 等,实际上都是一种混合型的语言,即在过程式的语言中加上面向对象的扩展。在 Java 中,几乎万物皆对象,就连一些基本数据类型,如整型、字符型、浮点型等,在 Java 中都可以作为对象处理。

Java 支持静态和动态风格的代码继承及重用,但不支持类似 C 语言那样的面向过程的程序设计。Java 语言的设计集中于对象及其接口,它提供了简单的类机制以及动态的接口模型。对象中封装了它的状态变量以及相应的方法,实现了模块化和信息隐藏。

1.2.3 分布式

Java 是一种面向网络的、支持分布式操作的程序设计语言。Java 包括一个支持 HTTP 和 FTP 的基于 TCP/IP 协议的子库。Java 应用程序可凭借 URL 类访问网络上的对象,其访问方式与访问本地文件系统几乎完全相同,使用非常方便。Java 的网络类库支持分布式的编程。Socket 类提供可靠的流式网络的连接,支持 TCP/IP 协议。通过编写协议句柄,程序员还可以扩充 Java 支持的协议集合。

在客户/服务器的模式下,Java 可以将运算从服务器端分散到客户端,提高系统的效率,避免了服务器的瓶颈制约。这些 Java 的特性使我们能够很容易地实现分布环境,尤其是为 Internet 提供动态内容。

1.2.4 健壮

健壮性又称为鲁棒性,鲁棒是英文 Robust 的音译,意思是健壮、强壮。Java 是一种非常健壮的语言,因为在 Java 中使用了以下手段:

- ◆ 不支持指针。在 C++ 程序中,指针的错误使用通常是程序中错误的元凶。在 Java 中彻底去掉了指针,杜绝了内存的非法访问,从而保证了程序的可靠性。
- ◆ 自动内存垃圾收集机制。Java 自动收集无用的内存单元进行内存管理,进而防止了由于内存泄漏导致的动态内存分配问题。Java 自己操纵内存减少了程序员在管理内存时出错的可能性。
- ◆ 完善的异常处理机制。Java 在编译和运行程序时,都要对可能出现的问题进行检查,以消除错误的产生。通过集成的面向对象的异常处理机制,在编译时 Java 提示出可能出现但未被处理的异常,帮助程序员正确地进行处理以防止系统的崩溃。Java 在编译时还可捕获类型声明中的许多常见错误,防止动态运行时不匹配问题的出现。除了检查程序在编译和运行时的错误,Java 还通过类型检查出开发初期出现的错误。这既简化了错误处理任务和恢复,也增加了程序的可读性。

1.2.5 与平台无关

网络是由很多类型的机器组成的,从 CPU 到操作系统都可能不相同。因此,使应用程序在各种机器上都能够无差别地执行,是人们长期以来追求的目标。因为 Java 最初是为对电子产品编程而设计的,所以它具有完美的平台无关性。

Java 将它的程序通过编译器编译成一种与平台无关的字节码文件格式,而不是通常的特定机器上的机器码,从而使 Java 的编译码能够在不同的平台运行。这些字节码对应于 Java 虚拟机中的表示,能够在平台上的 Java 虚拟机上运行,只要安装了 Java 运行系统,Java 程序就可在任意的处理器上运行。

1.2.6 安全

由于 Java 应用于网络程序的开发,因而安全性至关重要。因为 Java 小程序需要下载到客户端解释执行,如果没有安全控制,就会给一些网络黑客以可乘之机,这对用户来说是非常危险的。所幸的是,Java 的安全机制可以有效地防止病

毒程序的产生、下载程序对本地文件系统的破坏,以及网络黑客窃取密码和入侵。Java 提供了非常有效的安全控制,Java 的安全性可从以下几个方面得到保证:

- ◆ Java 的内存分配模式可以有效地防止有害代码的入侵。Java 摒弃了指针和释放内存等 C++ 功能,避免了非法的内存操作。由于取消了指针,一切对内存的访问都必须通过对象的实例变量来实现。因此防止了黑客使用特洛伊木马等欺骗手段访问对象的私有成员,也使得自己所建的程序无法访问它不应该访问的内存。另外,Java 的编译器不处理内存布局,运行时的实际内存布局无法从类的结构推知,这也同样提高了 Java 的安全性。
- ◆ 字节码验证器(ByteCode Verifier)可以防止 Java 运行环境装入有害的类文件,任何字节码都要通过字节码验证器来检验。这样,黑客通过炮制字节码来入侵的办法就行不通了。
- ◆ 浏览器限制小程序访问客户机的资源。小程序能够访问的客户机的资源比应用程序要少的多。浏览器有一个安全管理器用于控制小程序对文件系统和网络的访问,使小程序不能进行本地或远程访问。
- ◆ Java 类装入机制通过将本机类与网络资源类的名称分开,以此来保持安全性。有时,黑客使用一种称为“伪装”的攻击方法。伪装就是建立与标准类同名的类,希望能够冒名顶替那些比较敏感的控制资源的类。而 Java 的类装入机制在调入类时总要对它进行检查,因此能够防止假冒类的装入。当需要装入标准类时,Java 首先搜索 Java 运行系统自己的类定义集。如果自己的类定义集中没有该标准类,Java 会按照 ClassPath 指定的路径搜索本地机器中文件系统的其他位置,如果还没有,才去搜索远程服务器。

1.2.7 可移植

Java 提供了良好的可移植性,使用 Java 作为编程语言,只要进行一次程序开发工作,所开发的程序不需要经过任何改动,便能在各种平台上运行。Java 通过多种方法使 Java 应用程序能够不依赖于具体的系统:

- ◆ Java 提供了一个用于访问底层操作系统功能的可扩展类库,实现了与不同平台的接口,使这些类库可以移植。
- ◆ Java 编译器是由 Java 语言实现的,而 Java 运行系统由标准 C 语言实现,这就使 Java 系统本身也具有可移植性。
- ◆ 与平台无关的特性使 Java 程序可以方便地被移植到网络上的不同机器上,这是 Java 应用程序便于移植的良好基础。

另外,通过定义独立于平台的基本数据类型及其运算,Java 数据得以在任何硬件平台上保持一致。如果基本数据类型设计依赖于具体实现,将为程序的移植带来很大不便。例如在 Windows 3.1 中整数为 16 位,在 Windows 95 中整数为 32 位,在 DEC Alpha 中整数为 64 位,在 Intel486 中为 32 位。通过采用基于国际标准的数据类型,Java 的原始数据类型在任何机器上都是一样的,例如整数总为 32 位。在任何 Java 解释器中,数据类型都是依据以上标准具体实现的。因为几乎目前所有的 CPU 都能支持以上数据类型、8~64 位整数格式的补码运算以及单双精度浮点运算,所以 Java 数据也具有可移植性。

1.2.8 解释型

Java 解释器能够直接对 Java 字节码进行解释执行。字节码本身携带了许多编译时信息,使得链接过程更加简单。如果解释器速度较快,Java 可以在运行时直接将目标代码翻译成机器指令。Sun 用直接解释器一秒钟内可调用 300000 个过程,翻译目标代码的速度与没什么区别。因此,Java 的这一特性可以大大节省程序员在程序编译执行上所花的时间,使程序员能够把精力集中到源程序的编写上。

1.2.9 高性能

Java 是一种高性能的语言。通常,健壮性、安全性、平台无关性等方面的提高,总是要以牺牲性能为代价,Java 也不例外。因为 Java 运行系统必须内嵌一个内存管理模块,所以 Java 的内存管理增加了运行系统的复杂性。这样,Java 程序的解释执行的效率也要低于直接执行编译后的源码的效率。

但是,Java 采用了一些很好的措施来弥补这些性能上的差距:

- ◆ 生成高效的字节码。Java 字节码的设计充分考虑了性能的因素,字节码的格式简单,解释器可以生成高效的机器码。和其他解释执行的语言如 Basic、TC 不同,Java 字节码能很容易地直接转换成对应于特定 CPU 的机器码,从而得到较高的性能。
- ◆ 提供了即时编译和嵌入 C 代码的可选方案,其中即时编译是指在运行时把字节码编译成机器码。
- ◆ Java 支持多线程,这也有助于提高性能。Java 提供了对多线程的语言级的接口,而且 Java 环境本身也是多线程的。

1.2.10 多线程

Java 对多线程有良好的支持,Java 提供了语言内置的多线程控制,简化了多线程应用程序的开发,还支持线程的同步控制。多线程技术使得可以在一个程序里可同时执行多个小任务,而且同步机制保证了对共享数据的正确操作,多线程技术还可以提高图形用户界面的交互性能。

通过使用多线程,程序设计者可以分别用不同的线程完成特定的行为,而不需要采用全局的事件循环机制。这样,通过 Java 很容易地就可以实现网络上的实时交互行为,达到更好的交互性能和实时控制性能。虽然实时控制性能还取决于操作系统本身,但是在开发难易程度和性能上都比单线程要好。

1.2.11 Java 的动态特性

Java 的动态特性是其面向对象设计方法的扩展。它允许程序动态地装入运行过程中所需要的类,这是 C++ 语言进行面向对象程序设计所无法实现的。在 C++ 程序设计过程中,当在类中增加一个实例变量或一种成员函数后,引用该类的所有子类都必须重新编译。

Java 的动态性主要表现在以下几个方面:

- ◆ Java 的类有运行时的表示。即使在运行时,程序也能辨别类之间的关系和类型信息,可以动态地从本地或网上把一个类链接到运行系统中。
- ◆ 后期联编。Java 的类在运行过程中动态的装载,因此 Java 可以在分布式的环境中动态地维护应用程序和 Java 类库之间的一致性。当类库升级后,应用程序无需重新编译,也一样可以利用新类库中新增的功能。
- ◆ 支持动态数据类型和动态协议。通过编写协议句柄,Java 可以支持新的、自定义的传输协议。通过编写内容句柄,Java 可以支持新的数据类型。

1.3 Java 与 C/C++ 的比较

Java 是从 C/C++ 发展而来的,很多方面都与其相似。C 语言是一种效率很高、灵活性很强的计算机语言,相当一部分商业软件是用 C 语言编写的。然而,它没有内置的 OOP(面向对象编程)支持。C++ 的语法同 C 类似,但它有了内置的 OOP 支持。C 和 C++ 都是非常流行的计算机语言,目前的大部分软件都是由 C 或 C++ 编写的。由于 Java 的语法与 C 和 C++ 语言的语法类似,因此可以很容

易地将 C 或 C++ 程序改写成 Java 程序。

Java 被设计成 C++ 形式,在变量声明、参数传递、操作符、流控制等方面,Java 使用了和 C++ 相同的传统,使熟悉 C++ 的程序员能很方便地进行编程。同时,Java 为了实现其简单、健壮、安全等特性,摒弃了 C++ 语言的许多功能。这些功能都是很容易引起错误的地方,让 Java 的语言功能更精练。Java 同时还增加了一些很有用的功能,使 Java 比 C++ 更加安全、强大、易用。

下面,我们从几个方面对 Java 与 C++ 进行比较。

◆ 全局变量

Java 程序中,不能在所有类之外定义全局变量,只能通过在一个类中定义公用、静态的变量来实现一个全局变量,通过这种方式 Java 对全局变量进行了更好的封装。例如:

```
class GlobalVar{  
    public static global_var;  
}
```

在类 GlobalVar 中定义变量 global_var 为 public static,使其他类可以访问和修改该变量。

而在 C 和 C++ 中,能在所有类之外定义的全局变量常常造成意想不到的错误。

◆ goto

Java 不支持 C/C++ 中的 goto 语句,而是通过异常处理语句 try、catch、final 等代替 C/C++ 中用 goto 来处理遇到错误时跳转的情况,使程序有更强的可读性和结构化。

◆ 指针

指针是 C/C++ 中最灵活,也是最容易产生错误的数据类型。由指针所进行的内存地址操作常会造成不可预知的错误。同时通过指针对某个内存地址进行显式类型转换后,可以访问一个 C++ 中的私有成员,从而破坏安全性。而 Java 对指针进行完全的控制,程序员不能直接进行任何指针操作,例如把整数转化为指针,或者通过指针释放某一内存地址等。同时,数组作为类在 Java 中实现,很好地弥补了数组访问越界这一 C/C++ 中不做检查的漏洞。

◆ 内存管理

再次释放已释放的内存块或未被分配的内存块,会造成系统的崩溃,而忘记释放不再使用的内存块也会逐渐耗尽系统资源。在 C 中,程序员通过库函数 malloc 和 free 来分配和释放内存,在 C++ 中则通过运算符 new 和 delete 来分配和释放内存。这些操作都容易引起内存错误。在 Java 中,所有的数据结构都是对象,通过运算符 new 为它们分配内存堆。通过 new 得到对象的处理权,而实际分

配给对象的内存可能随程序运行而改变,Java 对此自动地进行管理并且进行垃圾收集。这种方式有效地防止了由于程序员的误操作而导致的错误,并且更好地利用了系统资源。

◆ 数据类型的支持

在 C/C++ 中,对于不同的平台,编译器对于简单数据类型如 int、float 等分别分配不同长度的字节数。例如: int 在 IBM PC 中为 16 位,在 VAX-11 中为 32 位,这导致了代码的不可移植性,但在 Java 中,对于这些数据类型总是分配固定长度的位数,如 int 型总占 32 位,这就保证了 Java 的平台无关性。

◆ 类型转换

在 C/C++ 中,可以通过指针进行任意的类型转换,这常常带来不安全性,而在 Java 中,运行系统对对象的处理要进行类型相容性检查,以防止不安全的转换。

◆ 头文件

C/C++ 中用头文件来声明类的原型以及全局变量、库函数等,在大的系统中,维护这些头文件是很困难的。而 Java 不支持头文件,类成员的类型和访问权限都封装在一个类中,运行系统对访问进行控制,防止对私有成员的操作。同时,Java 中用 import 语句来与其他类进行通信,以使用它们的方法。

◆ 结构和联合

C/C++ 中的结构和联合中所有成员均为公有,这就带来了安全性问题。Java 中不包含结构和联合,所有的内容都封装在类中。

◆ 预处理

C/C++ 中用宏定义来实现的代码给程序的可读性带来了困难。Java 不支持宏,它通过关键字 final 来声明一个常量,以实现宏定义中广泛使用的常量定义。

◆ 多重继承

多重继承是 C++ 中功能很强,但也难以掌握的内容。Java 取消了多重继承,但可以通过接口方式达到部分多重继承的作用。

◆ 操作符重载

操作符重载是 C++ 中强调程序可读性和便于理解的功能,但事实上经常造成混淆,所以 Java 不支持操作符重载。

1.4 为什么用 Visual J++ 进行 Java 编程

前面已经介绍了关于 Java 的一些背景知识,在了解了 Java 的众多过人之处之后,您也许在想自己动手的事情了——“现在开发工具这么多,我该选用什么

开发工具进行 Java 编程呢?”。

现在 Java 的开发工具有很多,具有代表性的有 Sun 公司的 JDK(Java Development Kits)、JWS(Java Workshop)和 Java Studio,IBM 的 Visual Age、Inprise 的 J++ Builder、Sybase 的 PowerJ、Microsoft 的 Visual J++、Symantec 的 Cafe 和 Visual Cafe,我国中科院软件所的 Java-Mate(Java 伴侣)。在本书中,我们除了介绍 Java 以外,还将介绍 Java 编程的利器——Microsoft 的 Visual J++,对于初学者来说这是进行 Java 编程的最好选择。我们将引导您进入 Visual J++ 的世界,您会发现用 Visual J++ 进行 Java 编程是如此的轻松、有趣。下面我们来看为什么选择用 Visual J++ 进行 Java 编程。

Visual J++ 是 Microsoft 公司推出的可视化的 Java 语言集成开发环境(IDE),为 Java 编程人员提供了一个新的开发环境,是一个相当出色的开发工具。无论集成性、编译速度、调试功能、还是易学易用性,都体现了 Microsoft 的一贯风格。在 Visual J++ 之前,Microsoft 公司在 Visual Developer Studio 集成开发环境基础上,已经推出了 Visual Basic、Visual C++、Visual FoxPro、Visual InterDev 等开发工具。Visual J++ 是 Microsoft 公司专门开发的面向 Java 的开发工具,和其他 Java 开发工具相比,Visual J++ 具有以下特点,这也是我们选择 Visual J++ 的理由。

- ◆ Visual J++ 把 Java 虚拟机(JVM)作为独立的操作系统组件放入 Windows,使之从浏览器中独立出来。这样,Microsoft 的其他产品,如 Word、Excel 和数据库都可以方便地使用 Java。
- ◆ Microsoft 的应用基本类库(AFC, Application Foundation Class Library)对 Sun 公司的 JDK(Java Development Kits)作了扩展,使应用基本类库更加适合在 Windows 下使用。
- ◆ Visual J++ 的编译器速度较快,每秒可以编译 10000 行以上的 Java 源程序。
- ◆ Visual J++ 的调试器支持动态调试,包括单步执行、设置断点、观察变量数值等。增强的调试支持还给集成的调试器加入了一些新的特征,包括多线程调试、远程调试、Java 和脚本的集成、用于调试脚本代码的运行文档窗口等。
- ◆ Visual J++ 提供了一些程序向导(Wizards)和生成器(Builders),如小程序向导、ActiveX 向导、数据库向导等,它们可以方便地帮助用户快速地生成 Java 程序,帮助您在自己的工程中创建和修改文件。例如,应用程序向导引导您用 WFC(Window Foundation Class)创建一个基于表单的应用程序。组件生成器为您的 WFC 组件添加属性和事件,数据表单向导给您的工程添加一个结合数据的表单,J/Direct 调用生成器(J/Direct Call Builder)向您的代码中自动加入 Win32 应用程序接口函数的 Java 定义。
- ◆ Visual J++ 界面友好,其代码编辑器具有智能感知、联机编译等功能,使程序编写十分方便。例如智能感知(IntelliSense),编程技术的这一集成能帮助

您快速、方便地编写代码。当键入代码时,代码窗口中在光标旁边会自动出现一个小窗口显示出成员列表和参数信息,只要在这个小窗口用鼠标选择想要键入的内容,语句就会很快被补充完整。

- ◆ Visual J++ 中建立了 Java 的 WFC,这一新的应用程序框架能够直接访问 Windows 应用程序接口(Windows API),使您能够用 Java 语言编写完全意义上的 Windows 应用程序。WFC 还能够把在 Internet Explorer 4.0 浏览器中执行的动态 HTML 模型打包,这样您在客户端和服务端就都可以动态操作 HTML。
- ◆ Visual J++ 中表单设计器的快速应用开发特性使用 WFC 创建基于表单的应用程序变得轻松、简单。运用 WFC 工具箱,可以快捷地将 WFC 控件拖放到表单上,并且可以很方便地在属性窗口配置这些控件的属性。
- ◆ Visual J++ 中通过 WFC 可以方便地使用 ActiveX 数据对象(ADO, ActiveX Data Objects)来检索数据和执行简单数据的绑定。通过在表单设计器中使用 ActiveX 数据对象,可以快速地在表单中访问和显示数据。
- ◆ Visual J++ 提供了增强的 COM 支持。在 Visual J++ 中创建 COM 对象是一个简单的过程,您还可以从 WFC 控件中创建 ActiveX 控件。
- ◆ Visual J++ 提供了对象浏览器。您可以通过对象浏览器,快捷地找到关于 Java 和基于 COM 的组件的信息,而不用把这些组件添加到您的工程中。
- ◆ Visual J++ 具有强大的打包和部署功能。您可以从工程或解决方案中对组件进行打包,并且把打包好的文件部署到 Web 上。
- ◆ Visual J++ 提供了多工程的解决方案。可以把一组工程编成一个解决方案,每一工程可以有一个不同的类型。例如,可以给同一个解决方案加入一个 Visual J++ 工程和一个 Visual InterDev 工程。
- ◆ Visual J++ 使用基于目录的工程。Visual J++ 工程根据文件系统被结构化地组织起来,工程中的每个文件和文件夹对应硬盘上的一个文件和文件夹,工程资源管理器帮助您管理工程中的文件和文件夹。
- ◆ Visual J++ 提供了全特征的 HTML 支持。HTML 编辑器能够帮助您在您的 Web 页中加入链接和书签,创建和修改 HTML 表格和脚本。HTML 编辑器还提供了 Web 页的三种视图,其中源代码视图(Source view)允许您编辑 HTML 源代码和标签,设计视图(Design view)为您编辑文件提供了一个所见即所得的环境,快速视图(Quick view)以相似于浏览器的风格显示您的 Web 页。

Visual J++ 的不足之处在于,用它开发的应用程序和小应用程序只能用于 Windows 平台。

第2章 快速熟悉 Visual J++

知识要点:

- ◆ 安装 Visual J++
- ◆ 集成开发环境的优点
- ◆ 熟悉 Visual J++ 的用户界面
- ◆ 创建第一个 Java 程序

在本章中,我们将从安装 Visual J++ 入手,快速熟悉 Visual J++ 集成开发环境的界面,了解 Visual J++ 集成开发环境的一些优点和主要特性,最后自己动手做一个简单的程序。



光盘 请参阅本书配套光盘的【基础知识】部分可交互学习与本章内容相关的知识。

2.1 安装 Visual J++

Visual J++ 很容易安装,为了使您的安装顺利进行,请仔细阅读本节,本节讨论了安装 Visual J++ 对计算机硬件的需求以及 Visual J++ 的安装步骤。

2.1.1 对硬件的要求

Visual J++ 对计算机硬件有较高的要求,下面列出了 Microsoft 指定的 Visual J++ 对硬件和操作系统的要求。

- ◆ CUP: 配有 Intel 奔腾处理器(Pentium 90 或更高)的 PC 机。
- ◆ 操作系统: Microsoft Windows 95/98 或更高版本,Windows NT Workstation 4.0 (带 Service Pack 3)或更高版本。
- ◆ 内存: Microsoft Windows 9x 下,24MB(建议 48MB);Windows NT 下,32MB(建议 48MB)。
- ◆ 硬盘空间: 典型安装 107MB;最大安装 157MB。
- ◆ 光驱: CD-ROM 驱动器。
- ◆ 显示器: VGA 以上(建议 Super VGA)。
- ◆ 外设: 鼠标或其他定点设备。

2.1.2 运行安装程序

首先在计算机中插入 Visual J++ 光盘,光盘中的安装程序将自动运行,出现安装引导程序窗口,如图 2.1 所示。如果该窗口没有出现,请打开资源管理器找到光驱根目录下的 Setup.exe 程序,用鼠标双击,执行这个程序。

在图 2.1 所示的安装引导程序窗口单击 Next 按钮,然后出现软件最终用户许可协议窗口(如图 2.2 所示)。选择 I accept the agreement(我接受该协议)后,单击 Next 按钮。

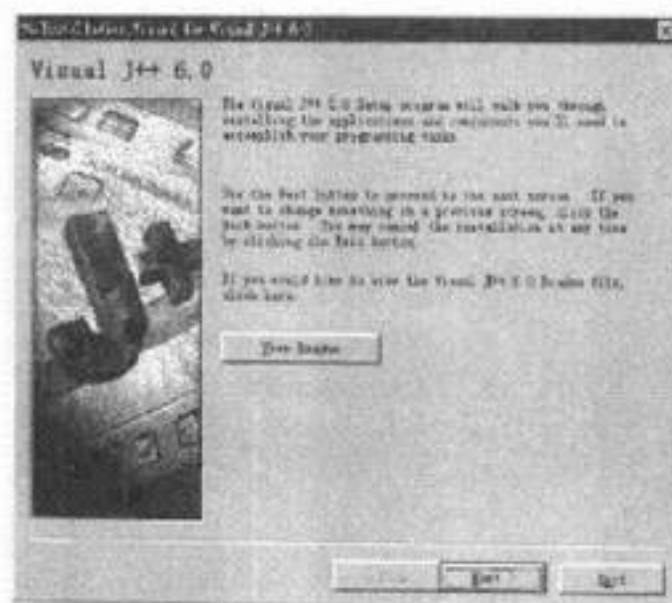


图 2.1 安装引导程序界面

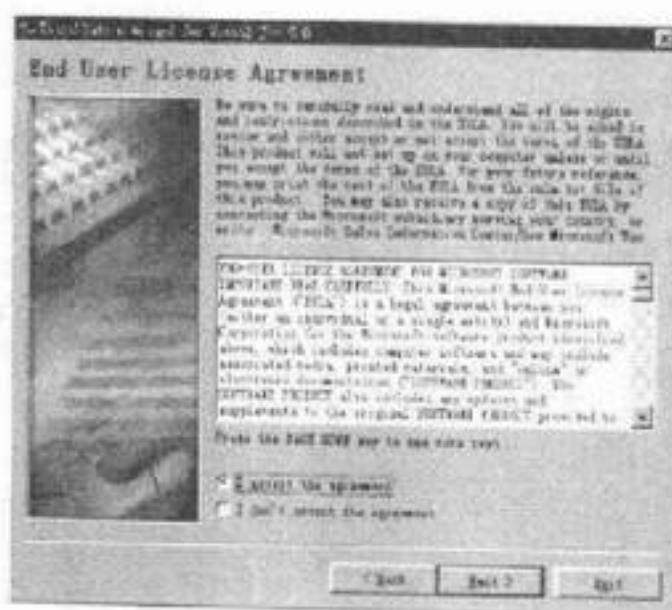


图 2.2 软件最终用户许可协议窗口

接受软件最终用户许可协议之后,进入产品注册窗口(如图 2.3 所示),在这里输入产品的 ID 号、姓名和公司的名字。其中姓名和公司的名字是可以任意填写的,但产品的 ID 号必须按照软件光盘所附带的资料填写。这里我们安装的是 Microsoft Press 版本,所以产品的 ID 号不必填写。

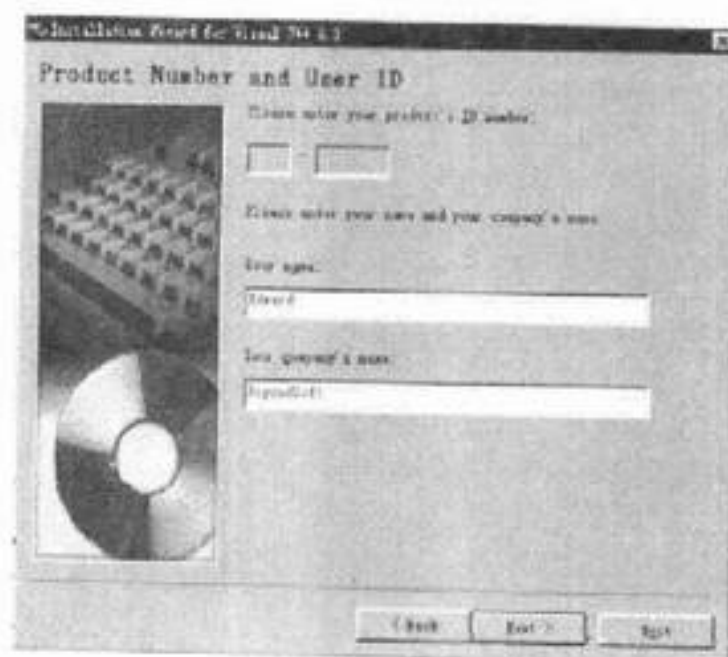


图 2.3 产品注册窗口

产品注册后的下一个窗口是安装项目选择窗口(如图 2.4 所示),这里有两个选项 Install Visual J++ 6.0(安装 Visual J++ 6.0)和 Server Application(服务器应用程序)。我们选择 Install Visual J++ 6.0,然后单击 Next 按钮。

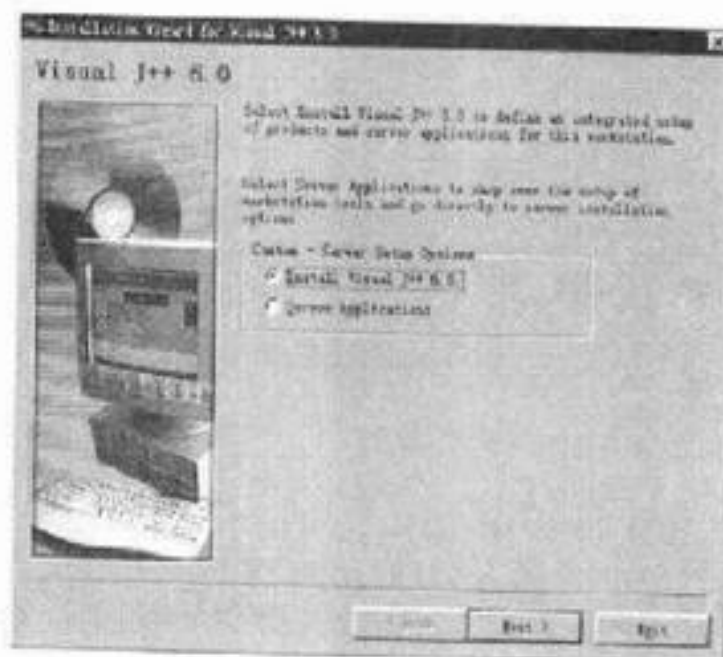


图 2.4 安装项目选择窗口

选择安装项目之后,下一步需要选择公共安装目录(如图 2.5 所示)。公共安装目录是用于存放 Visual Studio 6.0 的公用文件的目录,默认为“C:\Program Files\Microsoft Visual Studio\Common”。这里不用改它,直接单击 Next 按钮。



图 2.5 选择公共安装目录窗口

选择完公共安装目录后,便会出现信息窗口(如图 2.6 所示),上面显示信息是:正在启动 Visual J++ 6.0 安装,请等待……

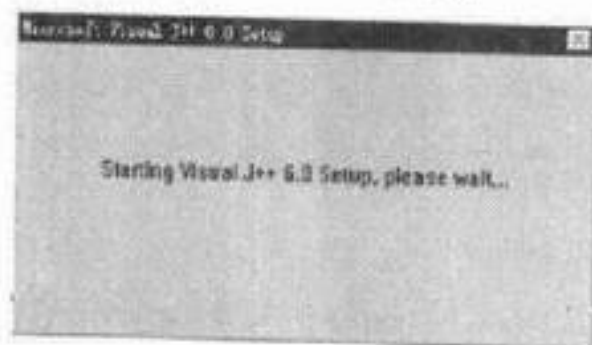


图 2.6 启动 Visual J++ 6.0 安装的信息窗口

接下来的两个窗口分别是 Visual J++ 6.0 的安装提示信息窗口(如图 2.7 所示)和产品 ID 号确认窗口(如图 2.8 所示),只要分别单击 Continue 按钮和 OK 按钮就可以了。



图 2.7 安装提示信息窗口



图 2.8 产品 ID 号确认窗口

确认完产品 ID 号之后,会出现如图 2.9 所示的窗口。在这个窗口,首先需要选择安装路径,这里默认的安装路径是“C:\Program Files\Microsoft Visual Studio\VJ98”,可以单击 Change Folder 按钮把默认的安装路径改变成其他路径。

选择完安装路径后,需要选择安装的类型:典型安装(Typical)或自定义安装(Custom)。典型安装将把 Visual J++ 6.0 所需的绝大多数组件复制到硬盘上,而自定义安装允许您自己指定要安装的组件。单击窗口左边相应图形按钮,就会开始典型安装或是自定义安装。



图 2.9 安装路径和安装类型选择窗口

接下来会出现一个信息窗口(如图 2.10 所示),所显示的信息为“安装正在检查所需的磁盘空间”。如果磁盘空间不足,安装程序会提示您清理磁盘或是删除一些其他文件,以便腾出 Visual J++ 6.0 的安装空间。

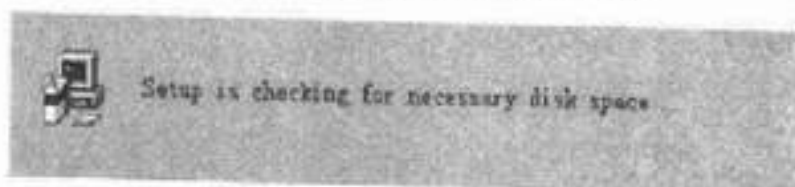


图 2.10 检查磁盘空间信息窗口

检查完磁盘空间之后,安装程序便开始从光盘向硬盘复制文件,这时会有一个安装进度指示窗口显示正在复制的文件名和安装的进度(如图 2.11 所示)。

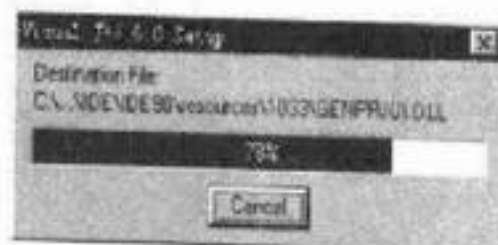


图 2.11 安装进度指示窗口

复制完文件后,会出现提示重新启动 Windows 的窗口(如图 2.12 所示),单击 Restart Windows 按钮,重新启动系统。

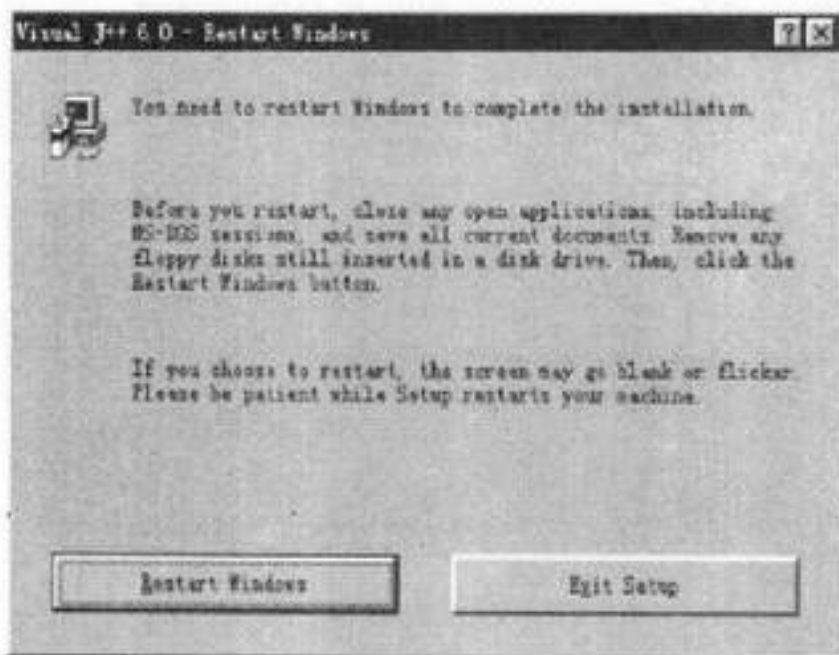


图 2.12 提示重新启动 Windows 的窗口

重新启动 Windows 之后,会出现一个窗口(如图 2.13 所示),提示您是否安装帮助文件,其默认的安装路径是“C:\Program Files\Microsoft Visual Studio\MSDN98”,单击左边的图形按钮进行安装。

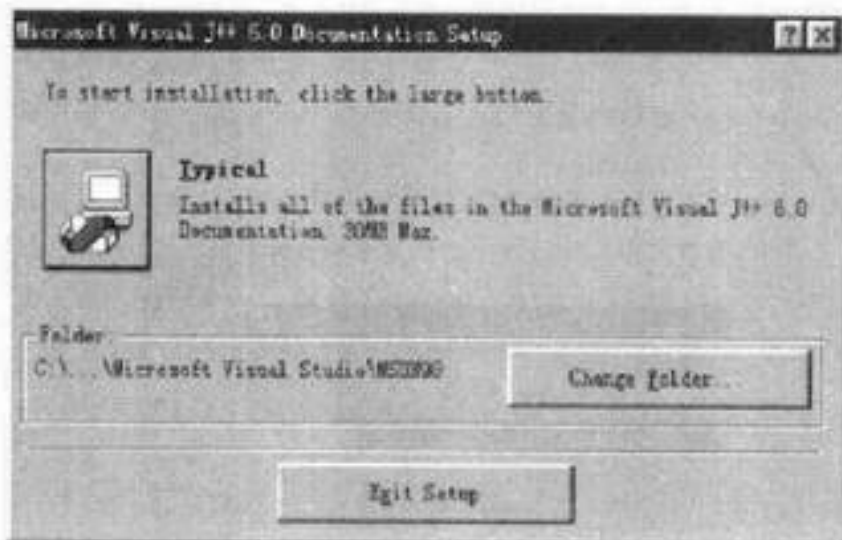


图 2.13 提示安装帮助文件窗口

安装完帮助文件后,还会有提示窗口提示是否安装服务器工具。服务器工具包括 NT Option Pack (for Windows 9x)、FrontPage 98 Server Extensions、Data Access Components 2.0 和 Visual J++ 6.0 Server Components 四项,可以选择一个或多个安装。我们现在不必安装这些服务器工具,直接单击 Exit 按钮就可以了。然后就会出现提示安装成功的信息窗口(如图 2.14 所示),单击 OK 按钮完成全部安装。

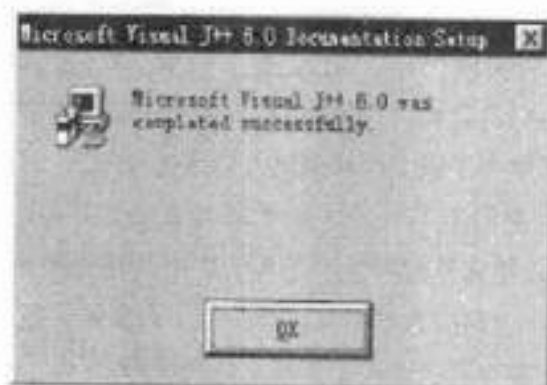


图 2.14 提示安装成功的窗口

2.2 集成开发环境的优点

Visual J++ 的集成开发环境允许您在单一的环境中创建、修改、编译、运行、调试和打包应用程序。

有人这样描述软件开发工具——“真正的程序员用 Visual C++, 机灵的程序员用 Visual Basic”, 这从某种角度说明了这两种开发工具的不同。众所周知, Visual C++ 是强有力的应用程序开发环境, 包括高级调试器、灵活的工程文件组建系统, 以及高度优化的编译器, 但是在快速开发应用程序方面相对比较繁杂; 而 Visual Basic 是一个快速的应用程序开发环境, 可以很快地开发出漂亮的界面, 但是调试、编译、灵活性方面相对于 Visual C++ 比较薄弱。我们这里学习的 Visual J++ 具备了上述两种语言各自的优点。

Visual J++ 的早期版本同 Visual C++ 的开发环境相似, 而现在的 Visual J++ 引入了 Visual Basic 的许多特征, 包括可视窗体设计和属性窗口。Visual J++ 继承了许多使 Visual Basic 语言得以成功的特点, 比如强有力的编程环境、高质量的代码生成功能、完善的调试功能等。这便是 Visual J++ 一体化的集成开发环境。

Visual J++ 集成开发环境有以下几个优点。

2.2.1 解决方案和工程文件系统

Visual J++ 在工程文件(Project)基础上又增加了一个解决方案(Solution)层。一个解决方案由一个或者一系列为了解决某个问题的工程文件所组成,而这些工程文件可以由不同的开发工具开发。例如一个解决方案可以由一个 Java 工程文件、一个 C++ 工程文件和一个 VB 工程文件等 3 个独立的工程文件组成。每个工程文件都有专门的元素,对要解决的问题全部或部分进行访问。集成开发环境的工作贯穿于整个工程文件和解决方案系统之中。

以这样的方式分别创建 3 个工程文件,就可以独立并灵活地应用它们。这种相对独立性,也可以使您及时对各个工程文件中的控件做出必要的改动。当然,最简单的情况是一个解决方案中只包含一个工程文件。在本书中,大部分的示例都是每个解决方案中只包含一个工程文件的情形。

2.2.2 Windows 基类 WFC 和 J/Direct

使用 Visual J++, 可以方便地利用和发挥 Windows 系统的优点,这是因为 Visual J++ 的集成开发环境提供了用于 Java 的 Windows 基类 WFC 和 Java 应用程序编程接口 J/Direct。Visual J++ 在以前的版本中,重点是创建 applet。但在 6.0 版中,Microsoft 将重点上升为将 Visual J++ 作为创建 Win32 应用程序的一个强大语言来使用。Windows 基类 WFC 框架及其组件模型就是 Visual J++ 实现这一目标的主要工具。

WFC 提供了简化 Windows 应用程序编写任务的类和接口,WFC 组件模型发挥 Java 的强大功能与稳定性,使您能够容易地创建 Windows 应用程序。WFC 还提供了操作动态 HTML (DHTML)的方法,DHTML 允许 Internet 开发者创建 Web 页和服务端应用程序。在用户从网上下载后,便可以对用户的 HTML 做相应的改动。

Java 依赖于包(Package)来处理大多数函数。WFC 在 com.ms.wfc.ui 包中,其中的绘图代码被进行了优化设计,更适合于 Win32 操作系统,如 Windows 95、Windows 98、Windows NT 操作系统的用户编程界面。它支持包括树视图、列表视图和多文本编辑等所有的 Win32 控件。

J/Direct 是 Java 为本机编程设计的应用程序编程接口。当 WFC 不能提供足够的支持时,可以利用 J/Direct API 访问包括 Win32 应用程序编程接口的 Win-

dows 动态链接库(DLL)。J/Direct 提供的是本机方法,属于非 Java 方法。这是因为它访问的是本地计算机,而不是 Java 虚拟机。J/Direct 的优点在于当 Java 虚拟机无法提供对给定的某个操作的支持时,可以使用本机方法。例如,可以使用 J/Direct 调用 Win32 API,以访问 Windows 注册表。另外,J/Direct 大大简化了 Windows 环境下 Java 程序的编写。

2.2.3 Internet 支持

Visual J++ 不仅对 Win32 应用程序的开发提供强有力的支持,而且在 Internet 方面,Visual J++ 使 Java Applet 的创建更为方便。实际上,Visual J++ 6.0 比它以前的版本具有更强的 Internet 开发功能。

Visual J++ 新的所见即所得的 HTML 编辑器允许您可视化地创建 Web 页,并可以直接将 Applet 拖放到 HTML 页上,也可以将 ActiveX 控件拖放到 HTML 页上。在创建使用脚本文件的 Web 页上,Visual J++ 提供了更强的 RAD 和脚本开发工具。并且当您要将 Applet 发布到 Web 上时,Visual J++ 的打包和部署工具将简化发布过程。Visual J++ 还提供了 DHTML 类库,您可以使用 Java 源代码来创建和控制 Web 页上的 DHTML 对象。

2.2.4 COM 支持

COM 是 Component Object Model(组件对象模型)的缩写,它是编写 Win32 应用程序的一项重要内容。不管是创建可重复使用的组件,还是创建多层客户/服务器应用程序,COM 都是一个必不可少的组件。Visual J++ 增加了 COM 新的 RAD 支持,其中包括 COM 对象的单击创建以及外部 COM 组件的引入等。

2.2.5 向导和生成器

Visual J++ 添加了一些向导和生成器来帮助您创建应用程序。如果想立即开始创建应用程序及其窗口和对话框,那么可以使用 Application Wizard(应用程序向导)。需要访问 WFC 框架外部的 Win32 应用程序编程接口(API)时,J/Direct Call Builder(调用生成器)可以为您创建正确的代码。为了帮助您开发和修改类和类成员,Visual J++ 提供了 Class Outline 窗口。需要为应用程序定制组件时,WFC Component Builder(WFC 组件生成器)可以使该过程流程化。

2.2.6 数据访问

为了简化数据库访问, Visual J++ 为 Java 数据库编程模型提供了强大的 ActiveX 数据对象(ADO), ADO 使 Java 对数据库的访问变得非常简单。为了更好地使用 Java 的 ADO, Visual J++ 提供了一些使用 WFC 的 Java ADO 组件。Java 的 ADO 还可以将 WFC 控件绑定到数据库的字段和记录中。

2.2.7 打包和部署

Visual J++ 的打包和部署两个功能也很有意义。打包功能可以将应用程序在其他系统中执行时所需要的重要文件组装在一起。因为所有的 Java 程序无非是类文件及其支持的源文件的一个集合, 所以 Visual J++ 的打包功能可以很容易地收集工程文件, 然后将它们组装成一个单独的文件, 使其他系统可以访问它们。可以使用打包功能来开发标识组包文件(.cab)或简单压缩文件(.zip), 以简化将 Applet 发布到 Web 上的过程。Visual J++ 打包工具在专业版和企业版都提供了打包功能。

创建了包含 Applet 或应用程序的数据包后, Visual J++ 的部署功能可以将应用程序或 Applet 发布到用户可以访问到的地方。无论工程是一个 Applet, 还是一个 Win32 应用程序, 都会发现部署功能对于发布项目是非常重要的。Visual J++ 提供了一个 Deployment Explorer(部署浏览器), 允许用户定义发布程序需要的所有信息。一旦完成了所有信息的定义, 就可以单击 Deployment Explorer 中的 Deploy 按钮, 然后用户的应用程序就可以被放到指定的位置。与打包工具不同, 部署工具只能在 Visual J++ 的企业版中获得(Visual J++ 的企业版随 Visual Studio 企业版一起发布)。

2.3 熟悉 Visual J++ 的用户界面

2.3.1 进入 Visual J++ 的用户界面

启动 Visual J++ 6.0 后, 屏幕上会出现 Visual J++ 6.0 的启动图像。之后出现 New Project 对话框(如图 2.15 所示), 可以通过这个对话框来创建新的 Visual J++ 工程文件, 也可以打开已有的工程文件。

如图 2.15 所示, 在左边的窗格中可以看到 Visual J++ Projects 文件夹, 在它

的下面是 Applications、Components 和 Web Pages 子文件夹,这里我们选择 Applications 子文件夹(这个子文件夹其实已经被默认选中了)。

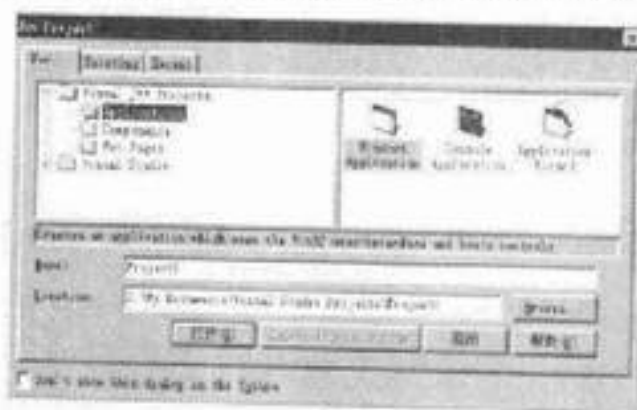


图 2.15 New Project 对话框

在右边的窗格里有 3 个图标,其标签分别为 Windows Application、Console Application 和 Application Wizard。我们选择 Windows Application,新建一个 Windows 应用程序。

在 Name 文本框中输入工程文件的名称(必须是合法的 Windows 文件名),这里我们对默认的工程文件名 Project1 不做修改。在 Location 文本框中输入合法的 Windows 目录名,工程文件将保存到此目录中。可以将工程文件保存在磁盘上的任意位置。这里,我们同样对默认的目录不做修改,将工程文件保存在该目录下。确定了工程文件的名称和保存位置后,单击【打开】按钮。

单击【打开】按钮之后,我们就在默认目录下新建了一个名为 Project1 的 Windows 应用程序。这时屏幕上会出现 Visual J++ 的集成开发环境(IDE),如图 2.16 所示。



图 2.16 Visual J++ 的集成开发环境

2.3.2 熟悉 Visual J++ 的用户界面

Visual J++ 提供了便捷、友好的用户界面,您可以在这个界面上方便地找到您所需要的工具,而且可以按您的习惯定义 Visual J++ 的用户界面。下面让我们从不同方面熟悉 Visual J++ 的用户界面。

菜单栏

如图 2.17 所示,菜单栏由 9 项菜单组成,分别是 File(文件)、Edit(编辑)、View(视图)、Project(工程)、Build(编译)、Debug(调试)、Tools(工具)、Window(窗口)和 Help(帮助)。



图 2.17 菜单栏

- ◆ **File(文件)菜单**: 提供文件操作的命令和选项,通过这个菜单可以新建工程文件、打开工程文件、关闭所有文件、增加工程文件、删除工程文件、新建文件、打开文件、关闭当前文件、保存文件、另存文件,还可以进行打印设置和页面设置。
- ◆ **Edit(编辑)**: 提供文档编辑的命令和选项,通过这个菜单可以选择全部文本、进行复制、粘贴、剪切、查找和替换等,还可以进行定义书签、插入文件等操作。
- ◆ **View(视图)**: 通过这个菜单可以打开工程管理器、属性窗口、工具箱、调试窗口等许多常用的窗口,还可以定义工具栏和窗口的布局。
- ◆ **Project(工程)**: 提供有关工程操作的命令和选项,通过这个菜单可以向工程文件加入窗体、Web 页、小程序等,还可以显示包、工程目录,进行部署、增加 COM 包装等操作。
- ◆ **Build(编译)**: 提供程序编译的命令和选项,通过这个菜单可以对程序进行编译、重编译、设置编译选项等。
- ◆ **Debug(调试)**: 提供程序调试的命令和选项,通过这个菜单可以完成运行程序、暂停运行程序、重新运行程序、运行到光标处、结束运行、单步跟踪调试、设置断点等调试操作。
- ◆ **Tools(工具)**: 提供文件操作的命令和选项,通过这个菜单提供 Visual J++ 的一些有用的工具,例如 WFC 组件生成器、J/Direct 调用生成器、插件管理器等,通过这个菜单可以对这些工具进行设置。
- ◆ **Window(窗口)**: 通过这个菜单可以完成相关的窗口操作,如层叠、平铺等。
- ◆ **Help(帮助)**: 提供 Visual J++ 的帮助文档。

工具栏



图 2.18 工具栏

工具栏(如图 2.18 所示)通过快捷按钮的方式把最常用的命令(如新建工程、打开文件、保存文件、复制、粘贴、运行、查找、工程管理器、属性窗口等)列在一栏中,把鼠标移到图标上停留一会儿,会出现该图标相应的说明。

代码编辑器



图 2.19 代码编辑器

代码编辑器(如图 2.19 所示)是编写程序的地方。Visual J++ 代码编辑器增加了许多重要的功能,使编写 Java 代码更加容易。最值得注意的是被称为“智能感知”(IntelliSense)的技术。“智能感知”在编写代码时提供帮助。例如,输入一个对象示例的名称,并输入一个句点来引用该对象的成员时,Visual J++ 会显示该对象的成员列表,可以进行选择,并将其插入到适当的位置。图 2.20 显示了“智能感知”的一个例子。

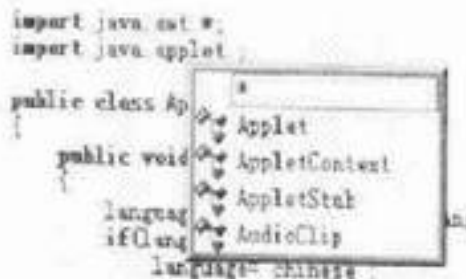


图 2.20 代码编辑器的“智能感知”

同样, Visual J++ 还通过方法参数来提供帮助。当引用一个方法时,它会在输入参数时显示该方法的原型。如果一个方法加载过多,那么 Visual J++ 允许滚动选择要引用的方法名称。图 2.21 显示了参数帮助的例子,当键入“concat”后,代码编辑器会自动弹出 concat()方法的使用规则,于是用户可以知道这个方法需要一个 String 类型的参数。

```
labelValue.concat(  
backgroundVString String concat (String p1)  
foregroundV
```

图 2.21 代码编辑器的参数帮助

Visual J++ 一个最有意义的方面是它的联机编译功能。类似于 Visual Basic 的代码编辑器, Visual J++ 可以在编译前检查代码的语法。当您在代码编辑器中输入代码时, Visual J++ 将检查编辑的代码行,并在发现错误的代码下显示一条红线。任务列表同时显示错误的解释说明。这种突出显示错误的方法类似于 Microsoft Word 在输入时的拼写检查。Visual J++ 的代码编辑器还提供了语法的颜色标识、断点管理,以及书签和快捷方式支持等功能。

工具箱

工具箱显示了可以添加到窗体中的组件和可以添加到 Web 页上的组件。只要用鼠标双击某个组件图标,这个组件就会自动添加到窗体或 Web 页上。或者用鼠标拖动某个组件直接把它放到窗体或 Web 页上。工具箱还可以帮助你组织将组件组织到工具箱的组中,以管理相关的组件。



图 2.22 工具箱

工程资源管理器

(Project Explorer)工程资源管理器是一个新的 Visual J++ 工具,它可以帮助您定

位和控制以 Visual J++ 目录为基础的工程。Project Explorer 可以显示工程目录和条目。使用 Project Explorer 可以添加或删除文件。使用它可以更容易地维护工程代码,确定正确的代码进行编译。Visual J++ 允许同时打开和使用多个工程。Project Explorer 会将工程组成解决方案,使用户可以容易地打开和使用相关的工程。

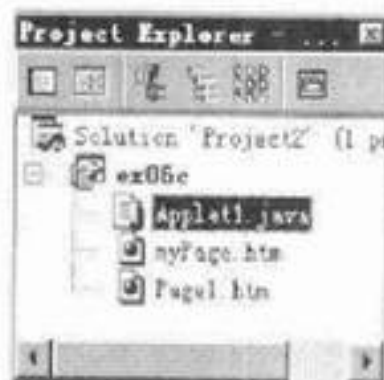


图 2.23 工程资源管理器

属性窗口

在 Project Explorer 窗口的下面,就是 Properties Browser(属性窗口),Properties 窗口显示了在窗体设计中选取的关于窗体、对象及控件的属性信息。Properties 窗口使您能够对窗体、对象及控件的属性进行设置。单击 Properties 窗口中的 Event 按钮(带有闪电图标),可以更改 Properties 窗口来显示窗体或组件的可用事件列表。在 Event View(事件视图)中,可以通过双击要处理的事件,或者输入要定义的事件处理程序方法的名称,来创建事件处理程序。图 2.24 显示的就是 DOCUMENT 对象的属性,可以在 Properties 窗口更改它的属性,例如背景颜色、宽度、标题等。



图 2.24 属性窗口

任务列表

任务列表(Task List)显示各种不同的信息。这些信息包括语法错误、编译错误、打包错误、TODO 注释和用户定义的任务等。Task List 最有吸引力的用途是创建代码编辑器中显示的用户定义的消息。通过包括带有 TODO 字样的、简单的 Java 注释,用户可以为其他用户添加注释。这些注释被 Visual J++ 读取,并放置在 Task List 中。双击列表中的 TODO 任务时,代码编辑器将切换到该注释所在的文件及相应的代码部分。使用 Task List 可以节省很多工作,因为它可以帮助您确定工程中尚未完成的任务;对于在不同时间编写同一段代码的程序开发小组来说,这是一项非常有用的功能。

图 2.25 显示出 3 条任务,两条是 TODO 注释,一条是语法错误。双击某一任务,光标就会停留在代码编辑器中相应的位置处。

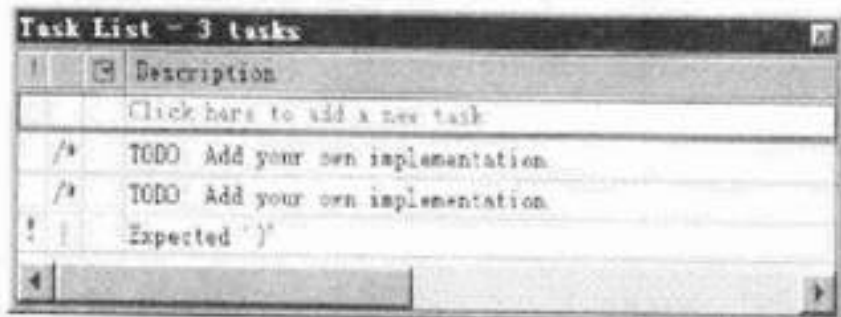


图 2.25 任务窗口

窗口布局

Visual J++ 集成开发环境还提供了一些其他窗口和工具,如 Class Outline 窗口、窗体设计器等,您可以随意调整各个窗口的位置和大小。这些窗口可以被单独放置在任何地方,也可以随时将窗口嵌在主窗口中或隐藏它。

如果想把窗口嵌在主窗口中,可以拖动窗口,使其靠近主窗口的边界,到一定程度它就会自动嵌在主窗口中。在拖动的过程中,只能改变窗口的位置,在完成停靠后,才可以改变窗口的大小。如果要避免窗口自动嵌在主窗口中,可以在拖动窗口时,按着 Ctrl 键。

还可以对窗口作一些特殊的配置。例如,可以在 View 菜单中选择 Define Window Layout 命令,在 Define Window Layout 对话框(如图 2.26 所示)中键入窗口布局的名称,单击 Add 按钮定义窗口布局(Window layout),以便能方便地返回到原来的窗口配置。

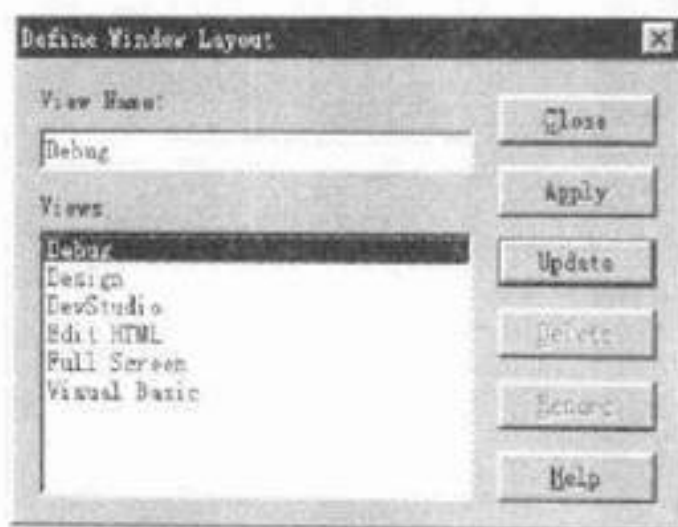


图 2.26 定义窗口布局

如果使用已经定义的窗口布局,可以在 Define Window Layout 对话框中选择想要的窗口布局,单击 Apply 按钮使用定义的窗口布局,单击 Close 按钮关闭对话框。

使用帮助

Visual J++ 的帮助系统非常庞大。可以在任何时候使用帮助,在编写 Visual J++ 程序时,有 4 种方式可以得到帮助:

- ◆ **F1 帮助。**这是最快的方式。只要单击一个单词、突出显示一条短语或单击一条错误信息,再按 F1 键就可以获得相应的帮助资料。可以在源代码编辑器窗口、帮助窗口或输出窗口中使用这种方式。
- ◆ **帮助索引。**帮助带有一个非常大的索引,包含绝大多数所需的信息。
- ◆ **搜索机制。**帮助是个联机系统,可以在帮助系统中搜索每条文本。搜索的范围比索引要广泛得多。
- ◆ **帮助目录。**帮助就像一本书一样也有目录,使用帮助窗口中的目录选项卡,可以阅读某个主题的所有内容。

Visual J++ 的帮助窗口如图 2.27 所示。Microsoft 使用 Internet Explorer 浏览技术显示帮助主题。可以使用帮助窗口左侧的目录、索引和搜索选项卡定位信息,右侧的窗口则显示相关主题。还可以使用帮助窗口工具栏上的前进和后退按钮浏览一系列主题。在帮助系统里迷路的时候,可以使用工具栏上的主页按钮回到帮助系统的起始点。

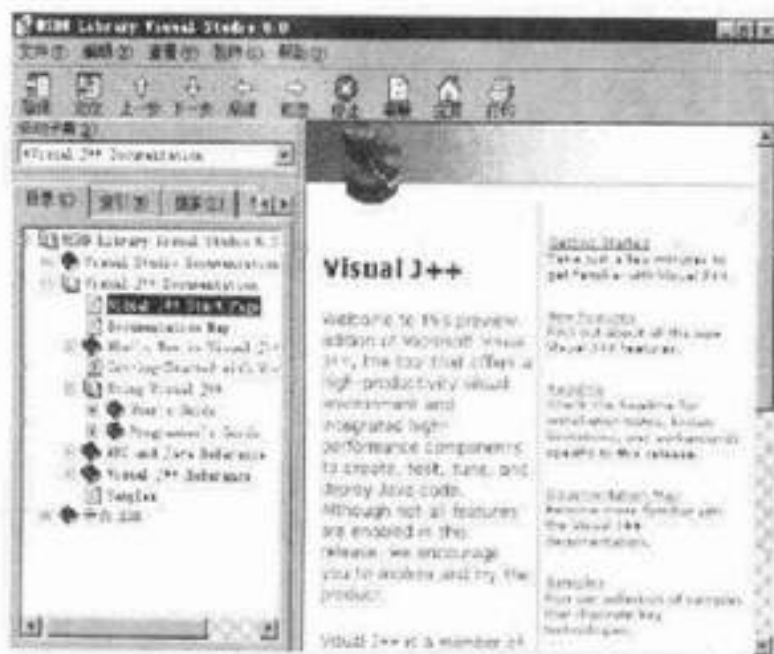


图 2.27 Visual J++ 的帮助窗口

2.4 从 Hello world 开始

下面我们通过编写一个显示“Hello world”的应用程序,使您对 Visual J++ 编程有个大概的认识。

2.4.1 创建工程和窗体

我们按照前面 2.3.1 小节所述的步骤,在目录 C:\My Documents\Visual Studio Projects\Project1 下新建一个名为 Project1 的 Windows 应用程序。这个 Windows 应用程序被称为是一个工程,所谓工程就是为了实现一定目标的一组文件的集合。

在集成开发环境(如图 2.28 所示)的工程资源管理器窗口(Project Explorer)中可以看到我们新建工程的名字——Project1,它以黑体字显示。工程名字上方有一个文件的图标,其标签为 Solution 'Project1' (1 project),这表示我们新建工程的同时也新建了一个名为 Project1 的解决方案。

在工程名字 Project1 的下方是一个文件的图标,其名称为 Form1.java,这表示工程 Project1 包含一个名为 Form1.java 的源文件。这个源文件对应一个名为 Form1 的窗体,在集成开发环境中间区域所显示的灰色、正方形窗口即为 Form1 窗体。我们就是要在在这个窗体上“写下”我们前面提到的“Hello world”。

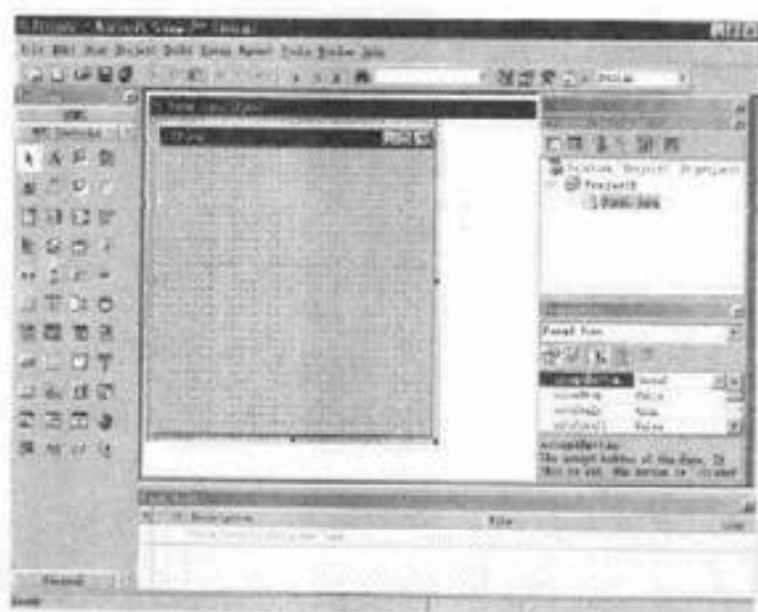


图 2.28 创建工程和窗体

2.4.2 添加显示的文本

首先,我们更改窗体的标题。在图 2.28 中,我们可以看到窗体 Form1 的标题栏显示的是“Form1”,我们想把它改成一个有意义的标题。在 Properties 窗口中,找到 text 属性(如图 2.29 所示),将“Form1”改为“我的第一个 Java 程序!”。这时我们会看到窗体 Form1 的标题栏显示为“我的第一个 Java 程序!”(如图 2.30 所示)。

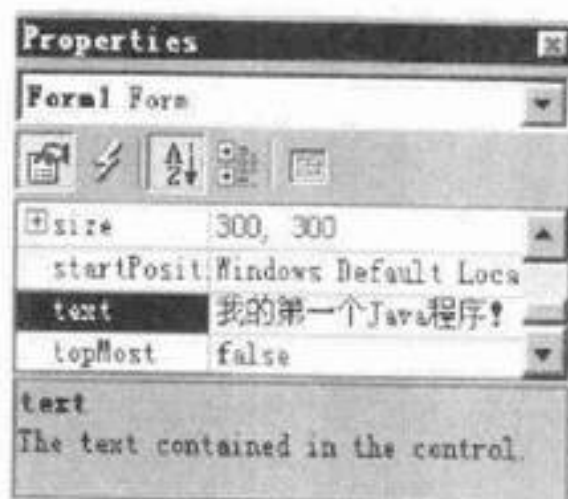


图 2.29 Properties 窗口



图 2.30 改变标题栏后的窗体

从 WFC Controls 工具箱(如图 2.31 所示)中找到一个标有大写字母 A 的按钮,这个按钮表示标签控件(Label),用于显示文本。双击这个按钮,这个标签控件就会自动添加到窗体中间(如图 2.32 所示)。可以改变这个控件的大小和位置。



图 2.31 WFC Controls 工具箱



图 2.32 向窗体上添加标签控件

下面我们来改变标签控件上的文字及其大小。选中刚才放在窗体上的标签控件,然后在 Properties 窗口中找到它的 text 属性,将其值“label1”改为“Hello world”,可以看到标签控件上的文字已经变成了“Hello world”。不过字显得有些小。选中窗体上的标签控件,再找到它的 font 属性,单击 font 左侧的标记(+),展开 font 属性,找到 font 属性下的 size 子属性,将其值修改为“16char”,可以看到标签控件上的文字已经变大了。



还可以通过另一种方式更改 font 属性。在 Properties 窗口选择 font 属性,这时 font 属性的取值栏的最右边出现一个  按钮,表示 font 属性是一个复合属性(包括子属性的属性)。单击  按钮,会弹出字体设置对话框(如图 2.33 所示),将字体设置为 Arial,字体样式为【**粗体**】,大小为 16,单击【确定】按钮结束设置。这时,我们可以看到标签控件上的文字字体变为粗体 Arial,结果如图 2.34 所示。



图 2.33 字体设置对话框



图 2.34 改变属性后的标签控件

2.4.3 运行程序

单击工具栏上的运行按钮(上有一个小箭头的按钮,鼠标移到上面会出现 Start 字样),运行程序,出现最后的运行结果(如图 2.35 所示)。



图 2.35 Hello world 程序结果

这样您就完成了简单的 Visual J++ 应用程序。



更上一层楼

在这个例子的基础上,您还可以进一步修改标签控件的其他属性,或者在窗体上添加别的控件,试试看会有什么新奇的结果。有的属性和控件可能您现在还不知该如何使用,不过随着您对 Java 语言和 Visual J++ 软件的了解,您会学到更多的知识,逐渐学会如何运用 Visual J++ 来开发 Java 程序。

在下一部分,我们将学习 Java 语言基础,了解关于 Java 语言的最基本的知识。

第3章 Java 编程概况

知识要点:

- ◆ 用命令行方式编译、运行 Java 程序
- ◆ Java 的程序结构
- ◆ 保留字、标识符
- ◆ Java 的数据类型,包括常量、变量的用法
- ◆ 数组和字符串
- ◆ 运算符、表达式
- ◆ Java 的流控制

在前面的章节中,我们对 Java 语言进行了概括性介绍,包括 Java 的起源与发展、语言特点、Visual J++ 的开发环境,相信您已经学会了如何使用 Visual J++ 的 Applet Wizard 创建一个小程序或独立的应用程序。但是,如果要进行深入的 Java 编程,就需要了解更多的、更详细的关于 Java 语言的知识。在接下来的章节中,我们将介绍 Java 语言的基础知识,Java 面向对象的特性和类、包、接口等内容。

本章介绍将 Java 语言的一些基本知识(不包括面向对象的概念)。我们将从命令行的 Java 应用程序开始,对如何用命令行方式编译、运行 Java 程序进行讨论。本章的大部分内容讲述了 Java 的基本语法,这是 Java 编程的基础。

现在您可以复习一下 Java 语言的背景材料,它的基本结构像 C/C++,但任何用面向过程语言编写过程序的人都可以了解 Java 语言的大部分结构。



光盘 参阅本书配套光盘的[基础知识]部分可交互学习与本章相关的知识。

3.1 程序结构

学习一种新的语言要从程序的结构开始,了解了程序的结构就了解了语言的基本组成,本节我们通过一个简单的例子来说明 Java 的程序结构。

3.1.1 命令行的“Hello World”Java 程序

学习一门新语言最好是先看一个简单的例子,所以我们从这个简单的 Java 程序入手来剖析 Java 程序的结构。运行 Visual J++, 打开 File 菜单,选择 New File 命令,如图 3.1 所示,选择 Java File 选项,单击【打开】按钮。

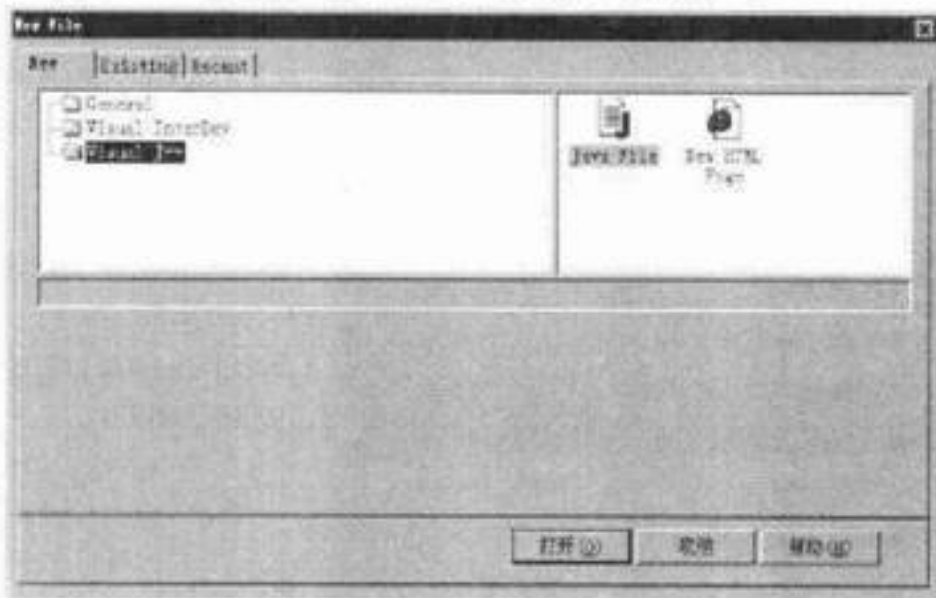


图 3.1 新建文件

在代码窗口中键入以下代码:

```
class ex03a {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

保存这个文件,建议使用和类名相同的文件名,即 ex03a.java。Java 源程序文件以 .java 为后缀,它是 Java 的编译单元,就如同在 C++ 中 .cpp 文件是其编译单元一样。当然也可以在任何文本编辑器中(比如 notepad)编辑上述文件,只要

把文件保存为 .java 为后缀文件。但是没有工程在 Visual J++ 环境下不能编译, 所以我们用 Jvc Java 编译器编译这个程序得到字节码文件(Bytecode), 字节码文件以 .class 为后缀, Jvc.exe 在 vj98(c:\Program Files\Microsoft Visual Studio\VJ98\)\目录下, 要在命令行窗口下进入此目录, 可键入如下命令:

```
Jvc.exe c:\JavaBook\Samples\Chap03\ex03a\ex03a.java
```

则在 C:\JavaBook\Samples\Chap03\ex03a 目录下生成字节码文件 ex03a.class。为解释执行 ex03a.class, 需要在命令行窗口运行 Jview, Jview.exe 通常在 Windows 目录下。可键入如下命令:

```
Jview c:\JavaBook\Samples\Chap03\ex03a\ex03a.class
```

程序将显示“Hello, World!”字符串。在这个极其简单的 Java 程序中, 有很多知识点。下面解释其中的每一部分:

- ◆ **class**: 这个关键字表示声明一个类。在 Java 中, Java 程序中的一切都以这样那样的类存在。所谓的类, 可以简单的理解为方法(函数)和数据的集合。
- ◆ **ex03a**: 紧跟着关键字 class 的是类名。编译器根据 Java 程序中的类的名字来给字节码文件命名, 而不是根据源程序文件的名字, 为了统一和便于识别, 应该使类名和源程序文件名相同。在这个例子中, 编译后得到的字节码文件为 ex03a.class。
- ◆ **static**: 如果您熟悉 C 或 C++, 那么对这个关键字一定很熟悉。如果您不知道 static 的作用, 暂且放过它, 在下一节中, 我们会给出详细的说明。这里, 只要把它看成程序中必不可少的一部分。
- ◆ **void**: 表示方法没有返回值。
- ◆ **main**: 方法名称。每个 Java 应用程序都需要一个 main 方法:

```
public static void main(String[] args)
{
    system.out.println("Hello, World!");
}
```

其中, 大括号表示了方法体的开始和结束。在这个方法中, 只有一条语句。可以根据自己的需要在方法中编写任意多的语句, 但是为了查看方便、结构清晰、易调试, 最好不要超过 30 行。在这条语句中, 使用了 system.out 对象, 调用它的 println 方法在控制台上输出一个字符串。在 Java 中, 字符串要用双引号括起来表示, 在语句的结尾, 要加上一个分号。

从这个简单的 Java 应用程序的例子中可以看出, 一个 Java 应用程序至少要包含以下两部分:

- ◆ 一个类, 这个类可以声明为 public, 也可以不声明。

- ◆ 一个 main 方法,其方法头必须写成如下格式:

```
public static void main(String arg[] args)
```

其中参数 args 用来接受命令行参数。

另外,Java 是区分大小写的,如 main 和 Main 代表两个不同的方法。而在有些语言中是不区分大小写的,例如 Pascal 语言。

一般来讲,Java 语言的源程序代码由一个或多个编译单元(compilationunit)组成,每个编译单元只能包含下列内容(空格和注释除外):

- ◆ 一个程序包语句(package statement)
- ◆ 入口语句(import statements)和类的声明(class declarations)
- ◆ 接口声明(interface declarations)

每个 Java 的编译单元可包含多个类或接口,但是每个编译单元最多只能有一个类或者接口是公共的(关于 Java 类、接口和包的详细内容见下一章)。Java 的源程序代码被编译后,产生 Java 字节代码。Java 字节代码是由不依赖于硬件的指令组成,这些指令能被 Java 虚拟机有效地解释执行。.class 文件便是 Java 虚拟机能够识别的代码。

3.1.2 注释

注释,就是程序中不会被编译执行的部分。它用来对程序代码作出解释、说明,以增强程序的可读性。如果您想暂且不执行某段代码,而又不想删除它,这时可以通过注释使这段代码无效。一名优秀的程序员总会在他的程序中加入必要的、简明扼要的注释。Java 中的注释主要有 3 种类型:

- ◆ //一行:表示从此处向后直到行尾都是注释
- ◆ /*一行或多行注释*/:在“/*”和“*/”之间的所有内容都是注释。当有大段的内容需要注释时,使用行注释就非常麻烦了,这时可以使用块注释。
- ◆ /** * 文档注释 */:一般放在一个变量或函数被定义的位置之前,在自动生成文档时被系统调用。

3.2 标识符

标识符是为了使变量、类、方法能被编译器识别而提供的名字,这个名字是不重复的。变量、函数、类和对象的名称都是标识符,在前面的例子中,main、ex03a 都是标识符。在 Java 语言里,标识符以字符_或 \$ 开头,后面可以包含数字。标识符

是区分大小写的,没有长度限制。Java 程序用 Unicode 编码,Unicode 字符集支持当今世界上的绝大多数的字符。这样就允许程序员在他们的程序中使用以他们的母语编写的字符,比如可以使用“自定义第一个类”作为一个类的类名。Java 中的方法 `Character.isJavaLetter` 可以判断一个字符是否存在于 Unicode 字符集。

下面给出了一些标识符的例子,其中有合法的,也有非法的。

◆ 合法标识符

```
mynameict_networkHello_sys_path$ bill, $199,getArea
```

◆ 非法标识符

```
5;0.4love,star *
```

相同的两个标识符具有相同的 Unicode 字符。但有时,看似相同的标识符也有可能是不同的。标识符不能使用 Java 的关键字、true 和 false 来命名。



注意 Unicode 是一种用以取代标准 ASCII 字符集的新编码标准。它把 ASCII 码由 8 位扩展成 16 位,以支持许多非拉丁语字符。Java 的 Char 和 String 类型数据采用的就是 Unicode 编码。

3.3 保留字

保留字是 Java 语言本身所使用的,不能另做它用。保留字也不能用作标识符,否则会产生编译错误。

表 3.1 列出了 Java 中所有的保留字,在以后的学习中,将会讲解每个保留字的含义。

表 3.1 Java 中的保留字

abstract	boolean	break	Byte	case
catch	char	class	Const	continue
default	do	double	Else	extends
final	finally	float	For	goto
if	implements	import	Instanceof	int
interface	long	native	New	package
private	protected	public	Return	short
static	super	switch	Synchronized	this
throw	throws	transient	Try	void
volatile	while			

下列是目前未使用的保留字: `cast`、`future`、`generic`、`inner`、`operator`、`outer`、`rest`、`var`。

3.4 数据类型

Java 是一种强类型语言,即 Java 中的每一个变量都必须有一个声明的类型。在 Java 中,有两种类型:基本类型和引用类型。基本类型有 8 种,分为数值型(numeric type)和布尔型(boolean type)两类。引用类型有 3 种,分别是类类型(class type)、接口类型(interface type)和数组类型(array type)。在本章中,引用类型只讲述数组类型,类类型和接口类型将在下一章中讲述。

注意,此外还有一种特殊的类型——`null` 类型。`null` 类型有唯一的值——`null` 引用。`null` 引用可以转换为任何引用类型,它实际上不引用任何对象。程序员可以忽略 `null` 类型,而把 `null` 假设为一个具有任意引用类型的文字。

3.4.1 常量

Java 中的常量值是用字符串表示的,分为不同的类型,如整型常量 123,实型常量 1.23,字符常量 'a',布尔常量 `true`、`false`,以及字符串常量 "his is a constant string。"与 C、C++ 不同,Java 中不能通过 `#define` 命令把一个标识符定义为常量,而是用关键字 `final` 来实现,如 `final double PI = 3.14159`。

3.4.2 变量

变量存储在内存中的某一位置。它用标识符标识,可以通过这个标识符来修改其中存储的值。变量有相应的类型,这个类型是编译时的类型。这种类型或者是基本类型,或者是引用类型。基本类型的变量总是拥有那个确定的基本类型的值。Java 中的所有类型,除了 `null` 类型以外,都可以定义变量。

变量的定义

变量的定义包括变量名、变量类型和作用域几个部分。在 Java 中,变量在使用之前必须要先定义。变量定义的格式如下:

数据类型 标识符[,标识符];

标识符是要定义的变量的名称,它由字母、数字、下划线或美元符 \$ 组成,Java 对变量名区分大小写,变量名不能以数字开头,而且不能为保留字。合法的

变量名如: myName、value_1、dollar \$ 等。非法的变量名如: 2mail、room #、class (保留字)等,变量名应具有一定的含义,以增加程序的可读性。变量类型可以为上面所说的任意一种数据类型。数据类型是指定义何种类型的变量,它必须是有效的 Java 类型;必须是 Java 中合法的标识符。方括号表示可选,在一条变量定义语句中,可以定义一个或多个变量,多个变量中间用逗号隔开。

变量的初始化

Java 程序中每个变量在使用前必须拥有一个值。Java 中不能引用没有值的变量,建立例子 ex03b,源码如下:



```
class ex03b
{
    static int x;
    public static void main(String[] args)
    {
        int x;
        int y = x;
    }
}
```

在命令行方式下编译此程序,会产生如下的编译错误:

```
Variable 'x' may be used before initialization
```

在 Java 中,变量在使用之前必须要先赋初值。可以在定义变量的时候赋给它初始值。例如:

```
int x = 3;
double dbl = 3.1415926D;
```

也可以在定义变量之后再对它赋值。例如:

```
int value;
.....
value = 1;
```

变量作用域

变量的作用域指明可访问该变量的一段代码。声明一个变量的同时也就指明了变量的作用域。按作用域来分,变量可以有下面几种:

- ◆ **局部变量。**在方法或方法的一块代码中声明,它的作用域为它所在的代码块(整个方法或方法中的某块代码)。

- ◆ **类变量。**在类中声明,而不是在类的某个方法中声明,它的作用域是整个类。
- ◆ **方法参数。**传递给方法,它的作用域就是这个方法。
- ◆ **例外处理参数。**传递给例外处理代码,它的作用域就是例外处理部分。

在一个确定的域中,变量名应该是唯一的。通常,一个域用大括号`{}`来划定。这里,我们只讲局部变量的作用域,即在块中定义的变量的作用域。块是指用一对大括号`{}`括起来的一系列语句和变量声明。比如,每个方法体都是一个块,在方法体内,也可以用`{}`定义块。块可以嵌套。例子 ex03c 说明了块的概念。



```
class ex03c
{
    public static void main(String[] args)
    {
        int body = 3;
        System.out.println("body=" + body + " 在方法体中");
        { //此处定义了一个子块
            int block1 = 4;
            System.out.println("block1=" + block1 + " 在第一层子块中");
            { //子块中又定义子块
                int block2 = 5;
                System.out.println("block2=" + block2 + " 在第二层子块中");
            }
        }
    }
}
```

程序执行结果如下:

```
body = 3 在方法体中
block1 = 4 在第一层子块中
block2 = 5 在第二层子块中
```

3.4.3 整型(integral type)

整型常量

与 C、C++ 相同,Java 的整常数有 3 种形式:

- ◆ **十进制整数。**如 123, -456, 0。

- ◆ 八进制整数。以 0 开头,如 0123 表示十进制数 83, -011 表示十进制数 -9。
- ◆ 十六进制整数。以 0x 或 0X 开头,如 0x123 表示十进制数 291, -0X12 表示十进制数 -18。

整型常量在机器中占 32 位,具有 int 型的值,对于 long 型值,则要在数字后加 L 或 l。例如: 324932432023L, 423432l。

整型变量

在 Java 中,有 4 种整数类型,如表 3.2 所示。

表 3.2 Java 中的整数类型

类型	存储空间大小
int	32 位
short	16 位
long	64 位
byte	8 位

int 是最常用的整数类型,它所表示的数据范围很大,比较适合于 32 位、64 位处理器;对于大型计算,常会遇到很大的整数,超出 int 类型所表示的范围,这时要使用 long 类型;当程序要求占用尽可能小的内存,同时要求较高的运行速度,这时,应多使用 short 类型;由于不同的机器对于多字节数据的存储方式不同,可能是从低字节向高字节存储,也可能是从高字节向低字节存储,这样,在分析网络协议或文件格式时,为了解决不同机器上的字节存储顺序问题,用 byte 类型来表示数据是合适的,而通常情况下,由于其表示的数据范围很小,容易造成溢出,应避免较多使用。

由于 Java 运行在虚拟机上,这样,整数类型便不依赖于具体的系统,即每种类型在任何一种机器上占用同样的存储空间,比如, int 总是 32 位, long 总是 64 位。这与 C++ 不一样, C++ 的整数类型的大小是和具体的机器有关的。

整型变量的定义

例如:

```
byte b; //指定变量 b 为 byte 型
short s; //指定变量 s 为 short 型
int i; //指定变量 i 为 int 型
long l; //指定变量 l 为 long 型
```


3.4.4 浮点型(float-pointtypes)

浮点型常量

Java 的实常数有两种表示形式:

- ◆ 十进制数形式。由数字和小数点组成,且必须有小数点,如“0.123”, “.123”, “123.”, “23.0”。
- ◆ 科学计数法形式。如: 123e3 或 123E3, 其中 e 或 E 之前必须有数字, 且 e 或 E 后面的指数必须为整数。

Java 中默认的浮点数是 double 型的, 如果要表示 float 型的浮点数, 可以使用后缀 F 或 f, 例如:

3.1434F, 9.343f

当然, 如果想要表示 double 型的浮点数, 还可以使用后缀 D 或 d, 例如:

3.141592653D, 1.33333333d

浮点型变量

浮点型变量用来表示带小数部分的数。在 Java 中, 有两种浮点类型变量, 单精度浮点型(float)和双精度浮点型(double), 如表 3.3 所示。

表 3.3 Java 中的浮点型

类型	存储空间大小	有效位数
Float	32 位	6~7
Double	64 位	15

双精度类型 double 比单精度类型 float 具有更高的精度和更大的表示范围, 更常使用。

实型变量定义

例如:

```
float f; //指定变量 f 为 float 型  
double d; //指定变量 d 为 double 型
```

3.4.5 布尔变量(Boolean)

布尔变量有两种取值: true 或 false。

3.4.6 字符型(char type)

字符常量

字符常量是用单引号括起来的一个字符,如'a','A','\t'(tab),'\u0000'(unicode)。

由于 Java 支持 Unicode 字符,故 Java 的字符型占用两个字节,用来兼容所有语言。Unicode 字符集是用来支持世界上的所有的语言的,ASCII 字符集只是它的一个子集,确切地说,它是 Unicode 字符集的前 255 个字符。所以,所有 ASCII 字符集中的字符都是 Unicode 字符集中的合法字符。Unicode 字符集的字符常常用十六进制表示,例如'\u0000'、'\u0028'、'\uffff'等。它的范围从'\u0000'到'\uffff',其中'\u0000'到'\u00ff'是 ASCII 字符集中的字符。'\u'是一个转义符号,用来表示紧接着的 4 个十六进制数字是 Unicode 字符的代码。其他常用转义字符如表 3.4。

表 3.4 Java 中的转义字符

转义序列	含义
\ddd	1 到 3 位 8 进制数所表示的字符(ddd)
\b	退格
\t	水平制表
\n	换行
\f	换页
\r	回车
\"	双引号
'	单引号
\\	反斜杠

字符型变量

字符型变量的类型为 char,它在机器中占 16 位,其范围为 0~65535。字符型变量的定义如:

```
char c = 'a'; //指定变量 c 为 char 型,且赋初值为'a'
```

与 C、C++ 不同,Java 中的字符型数据不能用作整数,因为 Java 不提供无符号整数类型。但是同样可以把它当作整数数据来操作。例如:

```
int three = 3;
```

```
char one = '1';  
char four = (char)(three + one); //four = '4'
```

上例中,在计算加法时,字符型变量 one 被转化为整数,进行相加,最后把结果又转化为字符型。

3.4.7 字符串

字符串是一些字符的序列。在 Java 中,没有字符串类型,字符串是通过 Java 类库中预定义的字符串类实现的。这个类叫做 String,它封装了字符串的大部分操作。有关类的详细内容见下一章,本小节我们从使用字符串的角度来讨论字符串。

字符串常量

与 C、C++ 相同,Java 的字符串是用双引号(")括起来的一串字符,如"This is a string.\n"。但不同的是,Java 中的字符串常量是作为 String 类的一个对象来处理的。

字符串常量是由双引号括起来的一个或多个字符的集合。在 Java 中,字符串文字采用对象化的办法,每个字符串都被处理成字符串对象。字符串的例子如下:

```
"我爱北京天安门","She's so vulnerable","我喜欢用 Java 语言编程"
```

字符串的定义

定义一个字符串,只要声明一个 String 类的对象即可。

例如:

```
String strFirstString = "JavaLanguage";  
String CompanyName = "Daphia";
```

常用的字符串操作有连接(Concatenation),取子串(Substrings)和比较。下面分别介绍。

连接

在 Java 中,可以使用"+"将两个字符串连接起来。例如:

```
String someOne = "我";  
String somewhat = "Java";  
String someAction = "喜欢";  
String sentence = someOne + someAction + somewhat;
```

经过连接后得到的字符串 `sentence` 为“我喜欢 Java”。在一个字符串中不能有回车（`'\u000a'`）和换行（`'\u000d'`），否则会产生编译错误。

当字符串和其他数据类型的值连接时，Java 中会先使用字符串转换将后者转换成字符串类型，然后再执行连接。在 Java 环境中，字符串转换允许把任何类型都转换成 `String` 类型。例如：

```
String str1 = "这是第";  
String str2 = "章";  
int chapcount = 3;  
String sentence = str1 + chapcount + str2;
```

得到的结果字符串 `sentence` 为“这是第 3 章”。Java 会自动地使用字符串转换将非字符串的值转换成字符串。

取子串

使用类 `String` 的方法 `substring`，可以从一个字符串中抽取一个子串。例如：

```
String fullPath = "c:\\Program Files\\DevStudio";  
String drive = fullPath.substring(0,1);
```

在这个例子中，我们从一个路径字符串中抽取驱动器字母，`drive` 的值为 `c`。在 Java 中，字符串第一个字母的位置是从 0 算起的，比如 `fullpath` 的第 0 个字符是 `c`，第 4 个字符是 `r`。

`substring` 方法的第一个参数是要抽取的子串的起始位置，第二个参数是您不想抽取的一串字符的第一个字符的位置。例如，要抽取第 0、1、2、3 个字符，那么第二个参数就应为 4。也就是说用第二个参数值减第一个参数值就得到了要抽取的子串的长度。

比较

在 Java 中，比较两个字符串是否相等，要使用 `equals` 方法。例如：

```
String s1 = "Internet";  
String s2 = "Intranet";
```

要比较这两个字符串是否相等，可以使用：

```
s1.equals(s2);
```

也可以使用：

```
s2.equals(s1);
```

当两个字符串相等时, `equals` 方法返回 `true`, 否则返回 `false`。在上例中, 返回值为 `false`。

比较字符串是否相等时不要使用操作符“`==`”, 此操作符只用来检查两个字符串是否在同一个存储位置, 换句话说, 操作符“`==`”可以用来检查两个字符串变量是否实现了存储共享。当两个字符串在同一个存储位置时, 使用“`==`”的比较结果为 `true`, 当它们不在同一个存储位置时, 即使它们内容相同, 比较的结果也是 `false`。实际上, 我们常常将同一个字符串的多个拷贝存储在不同的位置。例如:

```
String Song = "javaandinternet";
if (Song.substring(0,4) == "java")...
```

上述判断条件可能为 `false`, 因为编译器可能会为 `javaandinternet` 和 `java` 分配不同的存储位置, 而实际上 `Song.substring(0,4)` 得到的字符串可能和字符串 `Love` 的内容是相同的。

字符串的其他有用的方法

下面给出了类 `String` 的其他一些常用方法, 以及它们的详细描述。

```
char charAt(int index) '返回在指定位置的字符'
int compareTo(String anotherString)
```

将此字符串和指定的另一个字符串进行比较, 比较将按照 Unicode 字符集的顺序。如果指定的字符串在此字符串后面, 返回正值; 如果这两个字符串相等, 返回值为 0; 如果指定的字符串在此字符串前面, 返回负值。

```
boolean endsWith(String suffix)
```

判断此字符串是否以给定的后缀结束。此函数在对 URL 的操作中特别有用。

```
boolean equalsIgnoreCase(String anotherString)
```

忽略大小写判断此字符串是否和指定的另一个字符串相等。

```
int indexOf(String str)
```

返回在此字符串中指定的字符串第一次出现的位置。如果指定的字符串没有在此字符串中出现, 返回 -1。

```
int indexOf(String str, int fromIndex)
```

返回在此字符串中指定的字符串第一次出现的位置, 从 `int` 型参数 `fromIndex`

指定的位置开始搜索。如果指定的字符串没有在此字符串中出现,返回-1。

```
int lastIndexOf(String str)
```

返回在此字符串中指定的字符串最后一次出现的位置。如果指定的字符串没有在此字符串中出现,返回-1。

```
int lastIndexOf(String str,int fromIndex)
```

返回在此字符串中指定的字符串最后一次出现的位置,从 int 型参数 fromIndex 指定的位置开始搜索。如果指定的字符串没有在此字符串中出现,返回-1。

```
int length()
```

返回字符串的长度。

```
String replace(char oldChar, char newChar)
```

将此串中出现的 oldChar 都替换成 newChar,并生成一个新串。

```
boolean startsWith(String prefix)
```

判断字符串是否以给定的前缀开始。

```
String toLowerCase()
```

将此串中的所有字母都变成小写并生成新串返回。

```
String toUpperCase()
```

将此串中的所有字母都变成大写并生成新串返回。

```
String trim()
```

去掉此串前面和后面的空格并生成新串返回。

3.4.8 数组

数组是 Java 中的一种引用类型。在 Java 中,数组是作为对象来处理的。数组是由相同类型的元素组成的集合,它的元素可以是基本类型,也可以是引用类型。和字符串对象不同,数组的元素可以改变,但是大小是不能改变的。如果需要可变长数组,可以使用 vector 对象。您可以定义任意类型的数组。如:

```
chars[]; // 这是字符型数组
int[] array; // 这是整型数组
```

还可以定义数组的数组,如:

```
int block[][];
```


请注意,Java 在数组的定义中并不为数组元素分配内存,因此[]中不用指出数组中元素的个数,即数组长度,而且对于上面定义的一个数组,也不能访问它的任何元素。所以必须为它分配内存空间,这时要用到运算符 new,其格式如下:

```
arrayName = new type[arraySize];
```

其中,arraySize 指出数组的长度。如:

```
intArray = new int[3];
```

为一个整型数组分配 3 个 int 型整数所占据的内存空间。

```
intTdArray = new int[2][3]
```

为二维数组分配空间。

在 Java 里,由于数组实际上是一个对象,所以数组有一个成员变量: length。可以用这个成员函数来查看任意数组的长度。

3.5 运算符和表达式

前面已经介绍了各种数据类型,以及相应的变量、文字。当我们要对这些变量、文字进行操作的时候,就要用到运算符。这些运算符和变量、文字组合起来就组成了表达式。按操作数的数目来分,可以有一元运算符(如 ++、--),二元运算符(如 +、>)和三元运算符(如?:),它们分别对应于一个、两个和三个操作数。对于一元运算符来说,可以有前缀表达式(如 ++i)和后缀表达式(如 i++);对于二元运算符来说,则采用中缀表达式(如 a + b)。按照运算符功能来分,基本的运算符有下面几类:

- ◆ 算术运算符(+、-、*、/、%、++、--)
- ◆ 关系运算符(>、<、>=、<=、==、!=)
- ◆ 布尔逻辑运算符(!、&&、||)
- ◆ 位运算符(>>、<<、>>>、&、|、^、~)
- ◆ 赋值运算符(=,及其扩展赋值运算符如 +=)
- ◆ 条件运算符(?:)
- ◆ 其他(包括分量运算符·,下标运算符[],实例运算符 instanceof,内存分配运算符 new,强制类型转换运算符(类型),方法调用运算符())等)

Java 提供了非常丰富的运算符,本节将分别进行介绍。在本节的最后,还要介绍这些运算符的优先级和结合性。

3.5.1 算术运算符

算术运算符作用于整型或浮点型数据,完成算术运算。

二元算术运算符

二元运算符“+”将两个操作数相加并产生它们的和,而二元操作符“-”将两个操作数相减并产生它们的差。二元运算符“+”、“-”的操作数必须是基本数值类型,否则将产生编译错误。二元算术运算符如表 3.5 所示。

表 3.5 二元算术运算符

运算符	用法	描述	结合性
+	op1 + op2	加	左
-	op1 - op2	减	左
*	op1 * op2	乘	左
/	op1 / op2	除	左
%	op1 % op2	取模(求余)	左

Java 对加运算符“+”进行了扩展,使它能够进行字符串的连接,如“abc”+“de”,得到串“abcde”。我们将在第 7 章中讲解。

与 C、C++ 不同,对取模运算符%来说,其操作数可以为浮点数,如 $37.2\%10=7.2$ 。

一元算术运算符

一元算术运算符如表 3.6 所示。

表 3.6 一元算术运算符

运算符	用法	描述	结合性
+	+ op	正值	右
-	- op	负值	右
++	++ op, op ++	加 1	右
--	-- op, op --	减 1	右

+ (一元加)要求操作数是基本数值类型,否则将产生编译错误。

前缀 ++ 和前缀 -- 表达式的结果是值,而不是变量。这个值是加 1 或减 1 后存入变量的值。前缀 ++ 和前缀 -- 要求操作数的类型是基本数值类型,否则将产生编译错误。此操作符将对操作数进行加减运算,在将运算结果写回变量时,会根据需要使用缩窄基本转换以适应存储要求。表达式值的类型是操作数

变量的数据类型,而不是运算后的基本数据类型。



注意 `i++` 在使用 `i` 之后,使 `i` 的值加 1,因此执行完 `i++` 后,整个表达式的值为 `i`,而 `i` 的值变为未操作前的 `i` 加上 1。`++i` 在使用 `i` 之前,使 `i` 的值加 1,因此执行完 `++i` 后,整个表达式和 `i` 的值均为未操作前的 `i` 加上 1。`i--` 与 `--i` 之间的区别也一样。

3.5.2 关系运算符

关系运算符用来比较两个值,返回布尔类型值 `true` 或 `false`。关系运算符都是二元运算符,如表 3.7 所示。关系运算的结果返回 `true` 或 `false`,而不是 C、C++ 中的 1 或 0。

关系运算符常与布尔逻辑运算符一起使用,作为流控制语句的判断条件。

例如:

```
if( a > b && b == c)
```

表 3.7 关系运算符

运算符	用法	返回 true 的情况	结合性
>	<code>op1 > op2</code>	<code>op1</code> 大于 <code>op2</code>	左
>=	<code>op1 >= op2</code>	<code>op1</code> 大于或等于 <code>op2</code>	左
<	<code>op1 < op2</code>	<code>op1</code> 小于 <code>op2</code>	左
<=	<code>op1 <= op2</code>	<code>op1</code> 小于或等于 <code>op2</code>	左
==	<code>op1 == op2</code>	<code>op1</code> 与 <code>op2</code> 相等	左
!=	<code>op1 != op2</code>	<code>op1</code> 与 <code>op2</code> 不等	左

3.5.3 布尔逻辑运算符

布尔运算符用来对布尔变量或者结果为布尔型的表达式进行逻辑运算,其结果也是布尔型。表 3.8 给出了 Java 中的布尔运算符及其计算结果。

表 3.8 布尔逻辑运算符

<code>op1</code>	<code>op2</code>	<code>op1 && op2</code>	<code>op1 op2</code>	<code>! op1</code>
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false



注意 &&、|| 为二元运算符,实现逻辑与、逻辑或。! 为一元运算符,实现逻辑非。

对于布尔逻辑运算,“!”为右结合,其他为左结合。先求出运算符左边的表达式的值,对于或运算如果为 true,则整个表达式的结果为 true,不必对运算符右边的表达式再进行运算;同样,对于与运算,如果左边表达式的值为 false,则不必对右边的表达式求值,整个表达式的结果为 false。

3.5.4 位运算符

表 3.9 给出了 Java 中的所有的位运算符及其含义。

表 3.9 Java 中的位运算符及其含义

运算符	用法	说明	结合性
~	~ p1	按位取反	右
&	p1 & p2	按位与	左
	p1 p2	按位或	左
^	p1 ^ p2	按位异或	左
>>	p1 >> p2	有符号的右移, p1 右移 p2 位	左
<<	p1 << p2	左移, p1 左移 p2 位	左
>>>	p1 >>> p2	无符号的右移, p1 右移 p2 位	左

按位取反的操作数类型必须是基本整数类型,否则会产生编译错误。

运算符 &、|、^ 的操作数类型也必须是基本整数类型,它们对两个操作数进行逻辑计算,并以计算后的数据类型作为表达式的结果类型。

移位运算符 >>、<<、>>> 的两个操作数也必须是基本整数类型,否则会产生编译错误。移位运算符向左或向右移动一个整型数的位,移位运算符是二进制运算符,第二个移位运算符是决定移动的位数。

3.5.5 表达式

表达式是运算符、文字、变量组合起来的式子(在后面讲到方法也可以作为表达式的一部分)。下面给出了一些合法的表达式的例子:

```
name = "魏伟"
k = x + y + z
x++
2 * 3 * 5 * 7
```

```
System.out.println()
```

表达式可以有一个结果,这个结果可以是值(value),也可以是变量(variable),它们是由组成表达式的运算对象的类型决定的。值相当于C语言中的左值,即只能出现在赋值运算符的左边的值,变量则相当于C语言中的右值,即可以出现在赋值运算符的右边,用来保存对象或其他类型的值。当表达式的结果是变量时,也常常需要其值,在这种情况下,通常用“表达式的值”的说法来代替“表达式的结果”。

表达式也可以没有结果,唯一的例子就是对无返回值的方法的调用表达式。例如:

```
System.out.println()
```

方法调用表达式也可以调用有返回值的方法。但在这种情况下,返回值被忽略了。

3.5.6 运算符的优先级

运算符的优先级是指当不同的运算符组成表达式的时候,运算符计算的顺序。优先级高的运算符先运算,优先级低的后计算。表3.10给出了Java中的运算符的优先级。

表 3.10 Java 中运算符的优先级(自上而下按由高到低的顺序排列)

.	[]	()	
++	--	!	-
*	/	%	
+	-		
<<	>>	>>>	
<	>	<=	>=
==	!=		
~			
&			
^			
&&			
?:			
=	op=		

在表3.10中,同一行的运算符具有相同的优先级。同优先级的运算符在运

算的时候按照结合性进行计算。Java 中的结合性分成两种：左结合和右结合。左结合的运算符，如 `+`、`*` 等，它们计算的时候按照从左到右的顺序，例如：

```
a + b + c
```

等价于：

```
(a + b) + c
```

单目运算符，如 `-`、`~` 是右结合的，它们计算的时候按照从右到左的顺序，例如：

```
--a
```

等价于：

```
-(~a)
```

赋值运算符也是右结合的，例如：

```
a = b = c
```

等价于：

```
a = (b = c)
```

运算符的优先级和结合性又多又繁，如果您不是很清楚，建议使用圆括号来改变运算的顺序。这样既能达到预期的结果，看起来又很清晰。最内层括号内的表达式最先被计算，然后逐层向外计算。

表达式计算的顺序遵循以下规则：

- ◆ 最左边的操作数最先计算
- ◆ 先计算操作数，然后执行运算符的运算
- ◆ 参数列表的计算顺序从左到右

3.6 Java 流控制

与 C、C++ 相同，Java 程序通过流控制来执行程序流，完成一定的任务。程序流是由若干个语句组成的。语句可以是一条语句（如 `c = a + b;`），也可以是用大括号 `{ }` 括起来的一个复合语句。

下面分别来介绍 Java 中的流控制语句，包括：

- ◆ 分支语句：if-else, switch

- ◆ 循环语句: for, while, do-while
- ◆ 标号和转移语句: return, break, continue

3.6.1 分支语句

分支语句提供了一种控制机制,使程序的执行可以跳过某些语句不执行,而转去执行特定的语句。

条件语句 if-else

if-else 语句根据判定条件的真假来执行预先设定的两种操作中的一种,它的格式为:

```
if(boolean-expression)
    statement1;
[else
    statement2;]
```

- ◆ 条件必须是结果类型为 boolean 型的表达式,而且要用一对圆括号括起来。
- ◆ 每个单一的语句后都必须有分号。
- ◆ 语句 statement1、statement2 可以为复合语句,这时要用大括号{}括起。建议对单一的语句也用大括号括起,这样程序的可读性会更强,而且有利于程序的扩充(可以在其中添加新的语句)。{}外面不加分号。
- ◆ else 子句是可选的。
- ◆ 若布尔表达式的值为 true,则程序执行 statement1,否则执行 statement2……
- ◆ if-else 语句的嵌套形式为:

```
if(expression1){
    statement1
}else if(expression2){
    statement2
}.....
}else if(expressionM){
    statementM
}else{
    statementN
}
```

else 子句不能单独作为语句使用,它必须和 if 配对使用。else 总是与离它最近的 if 配对。可以通过使用大括号{}来改变配对关系。

多分支语句 switch

当处理多个条件时,可以使用分支语句 switch。分支语句根据给定条件的结果值进行判断,然后决定执行多路分支程序段中的哪一支。分支语句的形式如下:

```
switch (expression){  
    case value1 : statement1;  
        break;  
    case value2 : statement2;  
        break;  
    .....  
    case valueN : statementN;  
        break;  
    [default : defaultStatement; ]  
}
```

表达式 expression 可以返回任一简单类型的值(如整型、实型、字符型),多分支语句把表达式返回的值与每个 case 子句中的值相比。如果匹配成功,则执行该 case 子句后的语句。

case 子句中的值 value1 必须是常量,而且所有 case 子句中的值应是不同的。

default 子句是可选的,而且可以在 switch 语句中的任何位置,但无论在哪里,一定是最后执行的。当表达式的值与任一 case 子句中的值都不匹配时,程序执行 default 后面的语句;如果表达式的值与任一 case 子句中的值都不匹配且没有 default 子句,则程序不作任何操作,而是直接跳出 switch 语句。

break 语句用来在执行完一个 case 分支后,使程序跳出 switch 语句,即终止 switch 语句的执行。因为 case 子句只是起到一个标号的作用,用来查找匹配的入口并从此处开始执行,对后面的 case 子句不再进行匹配,而是直接执行其后的语句序列,因此应该在每个 case 分支后,用 break 来终止后面的 case 分支语句的执行。在一些特殊情况下,多个不同的 case 值要执行一组相同的操作,这时可以不用 break。

case 分支中包括多个执行语句时,可以不用大括号{}括起。

3.6.2 循环语句

循环语句的作用是反复执行一段代码,直到满足终止循环的条件为止,一个循环一般应包括4部分内容。

- ◆ 初始化部分 (initialization): 用来设置循环的一些初始条件,如计数器清零等。
- ◆ 循环体部分 (body): 反复循环的一段代码,可以是一条语句,也可以是复合语句。
- ◆ 迭代部分 (iteration): 在当前循环结束,下一次循环开始前执行的语句,常常用来使计数器加 1 或减 1。
- ◆ 终止部分 (termination): 通常是一个布尔表达式,每一次循环都要对该表达式求值,以验证是否满足循环终止条件。

在 Java 中,循环语句有 3 种形式:

- ◆ for 循环
- ◆ while 循环
- ◆ do...while 循环

for 循环

for 循环语句的一般形式如下:

```
for (initialization; termination; iteration) {  
    body;  
}
```

其中,括号中的 initialization 初始化语句、termination 判断表达式、iteration 更新语句都是可选的,如下的 for 循环语句是合法的:

```
for(;;) 语句
```

其中初始化语句只在循环开始的被执行,它可以只有一个表达式,也可以有用“,”分隔开的多个表达式,还可以有变量的声明和初始化。如果初始化语句有多个表达式,就按照从左到右的顺序逐个执行。通常初始化语句用来定义循环控制变量并置初值。

判断表达式的结果必须是 boolean 类型的,它决定了何时退出循环。如果表达式的值为 true,则继续下一次循环;否则,退出循环。如果没有判断表达式,则一直循环下去,除非遇到 break 语句才退出。

更新语句在每次循环的末尾执行,它通常用来改变循环变量。更新语句后不能加“;”。

while 循环

while 循环的一般形式是:

```

[initialization]
while (termination){
    body;
    [iteration;]
}

```

逻辑表达式的值必须是 boolean 型,否则将产生编译错误。

while 循环的执行过程如下:首先,计算逻辑表达式,如果表达式的值为 true,就进入循环,否则不能进入循环。每次执行循环体之前,要计算逻辑表达式,进行条件判断,一旦逻辑表达式的值为 false,就退出循环。

do...while 循环

do...while 循环的一般形式是:

```

[initialization]
do {
    body;
    [iteration;]
} while (termination);

```

和 for 循环、while 循环不同,do...while 循环先执行循环体,然后再计算逻辑表达式,进行条件判断。一旦逻辑表达式的值为 false,就退出循环,否则继续循环下去。究竟使用 do...while 循环,还是 while 循环,视具体程序的要求而定。

3.6.3 标号和转移语句

在使用转移语句,例如 break 和 continue 时,经常会用到标号语句。在 Java 中,一个代码块通常是用大括号 {} 括起来的一段代码。标号语句的一般形式如下:

```
BlockLabel: { codeBlock }
```

其中为 BlockLabel 标识符,作为标号只能出现一次,否则会产生编译错误。但是标号可以和变量名、方法名等重复,不会将变量、方法等隐藏。

Java 中没有 goto 语句,它有 3 种无条件转移语句: return、break、continue。

return 语句

return 语句用来从一个方法中返回。return 语句的一般形式为:

```
return;
```

或者

```
return 表达式;
```

`return` 语句从当前方法中退出, 返回到调用该方法的语句处, 并从紧跟该语句的下一条语句继续执行程序。前一种形式用在没有返回值的方法中, 后一种形式用于有返回值的方法中, 返回值的数据类型必须和方法声明中的返回值类型一致。可以使用强制类型转换来使类型一致。在 Java 中, 单独的 `return` 语句用在一个方法体的中间时, 会产生编译错误, 因为这时会有一些语句执行不到。但可以通过把 `return` 语句嵌入某些语句 (如 `if-else`) 来使程序在未执行完方法中的所有语句时退出, 例如:



```
int method (int num) {
// return num; //will cause compile time error
if (num > 0)
return num;
..... // may or may not be executed
//depending on the value of num
```

break 语句

`break` 语句的一般形式如下:

```
break;
```

或者

```
break BlockLabel;
```

`break` 语句第一种使用情况用来将控制权从一个封闭的语句中转移出来, 这个封闭的语句可以是 `for` 循环语句、`while` 循环语句、`do...while` 循环语句、`switch` 语句。

`break` 语句的第二种使用情况就是跳出它所指定的块, 并从紧跟该块的第一条语句处执行。

continue 语句

`continue` 语句必须出现在循环语句中, 它的作用是跳出循环。`continue` 语句的一般形式如下:

```
continue;
```

或者

```
continue outerLabel;
```

第一种 `continue` 语句用来结束本次循环,跳过循环体中下面尚未执行的语句,接着进行终止条件的判断,以决定是否继续循环。当控制权转移到 `for` 循环语句时,会进行循环条件测试,并执行更新语句;当控制权转移到 `while` 循环或者 `do...while` 循环时,直接进行循环条件测试。第二种 `continue` 语句将控制权转移到标号指定的封闭语句,然后这条语句立即结束当前循环并测试循环条件以决定是否开始新的循环。



更上一层楼

在本章中,系统地介绍了 Java 语言的一些基本知识(不包括面向对象的概念);从一个最简单的 Java 程序入手,通过剖析其结构,引入对程序的基本元素的介绍;介绍了 `java` 数据类型、表达式、运算符、流控语句。在下一章中,我们将学习 Java 的面向对象的概念、类、接口和包。

对于 Hello 程序,读者可以改变响应的文字,或运用本章学习的内容循环显示、根据不同的条件显示不同的内容。充分发挥您的想象力,可以把这个程序设计得更具个性。

第4章 Java 类、接口、包

知识要点:

- ◆ 面向对象编程的基本概念
- ◆ 如何定义 Java 的类
- ◆ 类与类之间的关系: 继承、重载、覆盖
- ◆ 如何创建和使用对象
- ◆ 抽象类和抽象方法的概念
- ◆ 如何定义、使用包
- ◆ 如何定义接口
- ◆ 在类中如何实现接口

第3章介绍了 Java 的简单数据类型、数组、字符串、运算符和表达式以及流控制方法。本章将介绍 Java 面向对象的特性以及面向对象编程的基本概念,然后讨论 Java 中的类、接口和包。



光盘 请参阅随书光盘的【基础知识】部分,可交互学习与本章内容相关的知识。

4.1 Java 类与对象

由于 Java 是一种面向对象的编程语言,所以在讲述 Java 类之前,简单介绍一下面向对象编程的概念。

4.1.1 面向对象编程的基本概念

面向对象方法是一种运用对象、类、继承、封装、聚合、消息传送、多态性等概念来构造系统的软件开发方法。

面向过程的编程语言,如 C、Pascal,是从解决问题的每一个步骤入手,它适合于解决比较小的简单问题。由于面向过程的编程语言几乎没有支持代码重用的语言结构,并且缺乏统一的接口,当程序的规模达到一定程度时,软件的开发将变得难以控制。

面向对象技术是从问题域中客观存在的事物出发来构造软件系统,用对象作为对这些事物的抽象表示,并以此作为系统的基本构成单位。每个对象都有其属性和服务,属性表示事物的静态特征,服务表示事物的动态特征从现实世界中客观存在的事物(即对象)出发来构造系统,并在系统构造中尽可能运用人类的自然思维方式。这种设计方法符合我们的思维习惯。

对象、类和消息

把具有相同属性和相同服务的对象归为一类,类是这些对象的抽象描述,每个对象是它的类的一个实例。通过在不同程度上运用抽象的原则,可以得到较一般的类和较特殊的类,特殊类继承一般类的属性和服务。对象的属性和服务结合为一体,对外屏蔽其内部细节,称作封装。复杂的对象可以用简单的对象作为其构成部分,称作聚合。对象之间的消息通信表达对象之间的动态联系,对象之间的关联表达对象之间的静态关系。

一个对象就是变量和相关的方法的集合,其中变量表明对象的状态,方法表明对象所具有的行为。例如:我们可以把汽车抽象为一个对象,用变量来表示它当前的状态,如速度、油量、型号、所处的位置等,它的行为则可以有加速、刹车、换挡等。操纵汽车时,不用去考虑汽车内部各个零件如何运作的细节,而只需根据汽车可能的行为使用相应的方法即可。实际上,面向对象的编程方法实现了对对象的封装,使我们不必关心对象的行为是如何实现的这样一些细节。通过对对象的封装,实现了模块化和信息隐藏,有利于程序的可移植性和安全

性,同时也利于对复杂对象的管理。

对象之间必须要进行交互来实现复杂的行为。例如,要使汽车加速,必须发给它一个消息,告诉它进行何种动作(这里是加速)以及实现这种动作所需的参数(这里是需要达到的速度等)。一个消息包含3个方面的内容:

- ◆ 消息的接收者
- ◆ 接收对象应采用的方法
- ◆ 方法所需要的参数

同时,接收消息的对象在执行相应的方法后,可能会给发送消息的对象返回一些信息。(如上例中,汽车的仪表上会出现已经达到的速度等)。由于任何一个对象的所有行为都可以用方法来描述,通过消息机制就可以完全实现对象之间的交互,同时,处于不同处理过程甚至不同主机的对象间也可以通过消息实现交互。

面向对象的程序设计语言中最重要的概念是类,在程序中,类主要有两个任务:

- ◆ 类定义了与其有关的抽象物。我们可以使用类描述真实世界的实体,无论它们有多复杂。
- ◆ 类是程序模块化的基础。

继承

通过对象、类我们实现了封装,而通过子类我们可以实现继承。使用继承,可以利用存在的类定义新类,而不用重新建立该类。子类可以从它的父类中继承元素和方法,对于上例来说,公共汽车、出租车、货车等都是汽车,但它们是不同的汽车,除了具有汽车的共性外,它们还具有自己的特点(如不同的操作方法,不同的用途等)。这时可以把它们作为汽车的子类来实现,它们继承父类(汽车)的所有状态和行为,同时增加自己的状态和行为。通过父类和子类实现类的层次,可以从最一般的类开始,逐步特殊化,定义一系列的子类。

同时,通过继承也实现了代码的重用,使程序的复杂性线性增长,而不是呈几何级数增长。在C++中支持多重继承,即一个类可以继承多个父类,这使对象的实现变得非常复杂且不可预料(设想多个父类拥有某些相同的变量和方法)。Java只支持单一继承,这大大降低了复杂度。在Java中通过接口可以实现多重继承,但接口的概念更简单,使用更方便,且不仅仅限于继承,它还使多个不相关的类可以具有相同的方法。

继承是一种功能非常强大的语言构造,但理解和使用它并不容易,下面是一些规则和技巧:

- ◆ 在理论中,继承代表了特殊化,可以把子类作为父类的一种特殊情况。
- ◆ 多重继承还可用于表示通用化。
- ◆ 实际上,继承是一种避免代码重复的手段,不必使用复制与粘贴技术建立两个相似的类,将一个类作为另一个类的父类将更加方便,而且可以节省调试时间。

多态

面向对象编程语言的第3个主要形式是多态,从字面上理解就是一个对象有多种形态。

在编程术语中,多态允许我们使用一个程序变量引用不同类的对象。Java通过方法重载来实现多态。通过方法重载,一个类中可以有多具有相同名字的方法,由传递给它们的不同个数和类型的参数来决定使用哪种方法,这就是多态。例如,对于一个作图的类,它有一个 draw() 方法用来画图或输出文字,可以传递给它一个字符串、一个矩形、一个圆形,甚至还可以重新指定作图的初始位置、图形的颜色等,对于每一种实现,只需定义一个新的 draw() 方法即可,而不需要新起一个名字,这样大大简化了方法的实现和调用,程序员和用户都不需要记住很多的方法名,只需要传入相应的参数即可。

通过方法重载,子类可以重新实现父类的某些方法,使其具有自己的特征。例如对于汽车类的加速方法,其子类(如赛车)中可能增加了一些新的部件来改善和提高加速性能,这时可以在赛车类中重载父类的加速方法。重载隐藏了父类的方法,使子类拥有自己的方法,这更进一步表明了与父类相比,子类所具有的特殊性。

4.1.2 Java 类

类是 Java 语言面向对象编程的基本元素,它定义了一个对象的结构和行为。在 Java 程序里,要表达的概念封装在某个类里。一个类定义了一个对象的结构和它的功能接口,功能接口称为成员函数或方法。在第3章的例子中,我们已经定义了一些简单的类,如 ex03a 类。类定义的一般形式如下:



```
class classname extends superclassname { typeinstance-variable1;
    typeinstance-variable2;
    .....
    typeinstance-variableN;
```

```

        typeMethodName1(parameter-list){
            method-body;
        }
        typeMethodName2(parameter-list){
            methodbody;
        }
        .....
        typeMethodNameN(parameter-list){
            method-body;
        }
    }

```

这里, `classname` 和 `superclassname` 是合法的标识符。类的定义可简单描述为:

```

classDeclaration {
    classBody
}

```

类定义的第一行是类声明,关键字 `extends` 用来表明 `classname` 是 `superclassname` 派生的子类。在 Java 中,有一个根类 `Object`,其他的类都是直接或间接从它派生而来。如果想定义 `Object` 的直接子类,可以省略 `extends` 子句,编译器会自动包含它。最外层大括号内为类体,类体中定义了该类所有的变量和该类所支持的方法。通常变量在方法前定义,类中所定义的变量和方法都是类的成员。对类的成员可以设定访问权限,来限定其他对象对它的访问,访问权限可以有以下几种: `private`, `protected`, `public`, `friendly`。同时,对类的成员来说,又可以分为实例成员和类成员两种。下面定义一个 `shape` 类。



```

class Shape {
}

```

随着内容的展开,我们也将扩展这个类。

对象实例

类名可以作为变量的类型来使用,如果一个变量的类型是某个类,那么它将指向这个类的实例,称为对象实例。所有对象实例和它们的类型都是兼容的。就像可以把 `byte` 型的值赋给 `int` 型的变量一样,您可以把 `Object` 的子类的任何实例赋给一个 `Object` 型的变量。

操作符 `new` 用来生成一个类的实例,下面这个例子生成了 `Shape` 类的一个

实例,存放在变量 `s` 中。



```
Shape s = new Shape();
```

在此例中,变量 `s` 指向这个对象,但并不真正包含这个对象。可以用多个变量指向同一个对象。下面的例子中,创建了一个 `Shape` 的对象,但创建了两个指向它的变量。

```
Shape s = new Shape(); s2 = s;
```

对 `s2` 所指向的对象的任何改动都会对 `s` 所指向的对象起作用,因为它们已经是同一个对象。对 `s` 和 `s2` 的赋值只是把它们指向这个对象,既没有分配内存,也没有复制这个对象的任何部分。对 `s` 的再赋值只是简单地断开了 `s` 和原来对象的联系,并不影响对象本身,下面的例子说明了这种情况。

```
Shape s = new Shape(); s2 = s; s = null;
```

尽管 `s` 被赋值为 `null`,`s2` 仍指向原来由操作符 `new` 创建的对象。在前面的例子里,生成了一个对象并且指向它两次。这就允许两个变量改变同一个对象。

一个实例是类模板的单独的拷贝,带有自己的称为实例变量的数据集。每个实例也可以作为一个对象。当定义一个变量的类型是某个类时,它的默认值是 `null`,`null` 是 `Object` 的一个实例。对象 `null` 没有值,它和整数 `0` 不同。下面这个例子中,声明变量 `s` 的类型是类 `shape`。

```
Shape s;
```

这里,变量 `s` 的值是 `null`。

创建一个新的对象时,可直接对它的实例变量赋值。每个对象都有它所属类的实例变量的副本,每个对象的实例变量都是和其他对象的实例变量分离的,所以改变一个对象的实例变量不会影响其他对象的实例变量。

修饰符

修饰符是影响变量及方法的生命期的关键字。表 4.1 列出了类、成员变量、局部变量和方法可用的各种修饰符。

修饰符分为存储与生命期修饰符和可访问修饰符。存储与生命期修饰符有: `abstract`、`static`、`synchronized`、`native`、`transient`、`volatile` 和 `final`。

可访问修饰符有: `public`、`protected` 和 `private`。

表 4.1 类、成员变量、局部变量和方法可用的修饰符

修饰符 P	类	方法	成员变量	局部变量
abstract(抽象)	✓	✓		
static(静态)	✓	✓	✓	
public(公用)	✓	✓	✓	
protected(保护)	✓	✓	✓	
private(专用)	✓	✓	✓	
synchronized(同步化)		✓		
native(本地的)		✓		
transient(过渡)			✓	
volatile(易失的)			✓	
final(最终的)	✓	✓	✓	✓

“✓”表示可以修饰。例如,abstract 可修饰类

存储与生命期修饰符

◆ abstract

abstract 类必须被继承,abstract 方法必须被重载。有时需要定义一个给出抽象结构,但不给出每个成员函数的完整实现所需的类。如果某个成员函数没有完整实现,必须要由子类来覆盖,可把它声明为 abstract 型。抽象方法,为所有子类定义一个统一的接口,对抽象方法只需声明,而不需实现,其格式如下:

```
abstract returnType abstractMethod([paramlist]);
```

抽象类中不一定要包含 abstract 方法,但是一旦某个类中包含了 abstract 方法,则这个类必须声明为 abstract 类。含有抽象型成员函数的类必须声明为抽象的。为了把一个类声明为抽象的,只需在类定义的 class 关键词前放置关键字 abstract。定义一个抽象类的格式如下:

```
abstract class abstractClass{
    .....
}
```

下面例子把 Shape 类定义为抽象类,Circle 为其子类,Circle 类必须实现 draw 方法。

```
abstract class Shape {
    Abstract void draw();
};

class Circle extends Shape {
```

```
void draw() {
    System.out.println("Circle draw");
}
```

当一个类的定义完全表示抽象的概念时,它不应被具体化为一个对象,而且它们的完整实现还没有定义,因此抽象类不能直接用 new 操作符生成实例。也不能定义抽象的构造函数或抽象的静态成员函数。抽象类的子类或者实现了它的超类的所有抽象的成员函数,或者也被声明为抽象的。由于抽象类不能被具体化,因此下面的语句会产生编译错误:

```
new Shape ( ); //abstract class can't be instantiated
```

◆ static

通常,类的每个实例都有自己的成员变量备份。但是,可以制定一个成员变量属于类本身,独立于类中的任何对象。这种成员变量称为 static(静态)成员。如果要创建一个可以在实例外部调用的成员函数,那么只需声明它为静态的,它就会正常运行。静态成员函数只能直接调用其他静态成员函数,而不能以任何方式使用 this 或 super。

◆ synchronized

同步化方法一次只让一个线程执行该函数,这防止了两个执行线程相互抵消彼此的工作。关于线程的概念见第 7 章。

同步方法默认为非静态的,但也可以声明为静态的。

同步化修饰符不适用于类或成员变量。

◆ native

native 方法用其他语言(如 c 语言)实现,所以没有程序代码块。核心 API 中的许多方法是 native,因为它们需要访问特定操作系统的过程,如在屏幕上画图。

native 修饰符只适用于方法。

◆ volatile

volatile 变量是数值可能不依靠 Java 程序本身而改变的变量。关键字 volatile 可防止编译器去跟踪该变量的改变情况。编译器假设每次 Java 程序访问都是给变量赋予了新值。

由多个并发线程共享的变量可以用 volatile 来修饰,使得各个线程对该变量的访问能保持一致。

◆ transient

transient 和序列化一起提供持续对象。

◆ final

在默认情况下,所有的成员函数和实例变量都可以被覆盖。但由于安全性的原因或者是面向对象的设计上的考虑,有时候希望一些类不能被继承,例如,Java 中的 String 类,它对编译器和解释器的正常运行有很重要的作用,不能对它轻易改变,因此把它修饰为 final 类,使它不能被继承,这就保证了 String 类型的唯一性,同时,如果你认为一个类的定义已经很完美,不需要再生成它的子类,这时也应把它修饰为 final 类,定义一个 final 类的格式如下:

```
class finalClassName|
.....
|
```

同样,有些方法不能被重载,这时把它限定为 final 方法,其格式为:

```
final returnType finalMethod ([paramlist]){
.....
|
```

可访问修饰符

◆ private

类中限定为 private 的成员只能被这个类本身访问。它的声明如下:

```
private privateVar;
private privateMethod ([paramlist]){
.....
|
```

同一个类的不同对象可以互相访问 private 成员变量或互相调用 private 方法,这是因为访问保护是在类的级别上,而不是对象的级别上。

另外,对于构造方法,也可以限定它为 private。如果一个类的构造方法声明为 private,则其他类不能生成该类的一个实例。

◆ protected

类中限定为 protected 的成员可以被这个类本身、它的子类(包括同一个包中以及不同包中的子类)以及同一个包中所有其他类所访问。它的声明如下:

```
protected protectedVar;
protected protectedMethod ([paramlist]){
.....
|
```

处在不同包中的子类虽然可以访问父类中限定为 protected 的成员,但这时访问这些成员的对象必须具有子类的类型或者是子类的子类的类型,而不能是

父类类型。

◆ public

类中限定为 public 的成员可以被所有的类访问。它的声明如下：

```
public publicVar;
public PublicMethod ([paramlist]) {
    .....
}
```

成员变量

类体中的变量叫做成员变量,最简单的成员变量的声明为:

```
type variableName;
```

成员变量的类型可以是 Java 中的任意数据类型,包括简单类型、数组、类和接口。在一个类中,成员变量应该是唯一的,但是成员变量的名字可以和类中某个方法的名字相同,例如:

```
class Shape {
    int area;
    int linewidth;
    int area() {
        return this.area;
    }
}
```

在这个例子中,分别定义了 area 成员变量和 area() 方法,area 成员变量用来表示 Shape 的面积大小,area 方法求 Shape 的面积,其中,area() 和变量 area 具有相同的名字。

类的成员变量和在方法中所声明的局部变量是不同的,成员变量的作用域是整个类,而局部变量的作用域只是方法内部。

成员函数

成员函数是一个方法,这个方法是类的功能接口,也可以说是类里定义的一个子程序。在类里的定义和实例变量处于同一级别。必须通过一个类的实例来调用成员函数。成员函数定义的一般形式如下:

```
type methodName (formal-parameter-list) {method-body;}
```

这里,type 指的是成员函数的返回值的类型,如果没有返回值,就用 void 类型。methodName 可以是任何合法的标识符,但不能与当前的类名相同。formal-

parameter-list 是类型、标识符的序列。如果没有参数,括号里就是空的。在类 Shape 中添加 getlinewidth 方法如下:

```
class Shape{
    int linewidth;
    int getlinewidth(){
        return this.linewidth;
    }
    int setlinewidth(int width){
        this.linewidth=width;
    }
}
```

因为每个成员函数都在类的个别实例内执行,而我们创建的类的实例具有它自己的实例变量,所以成员函数可直接使用它们。

成员函数调用可以用点“.”操作符来调用一个类的实例的成员函数。成员函数调用的一般形式如下:

```
objectreference.methodname(parameter-list);
```

这里,objectreference 是指向某个对象的变量,methodname 是 objectreference 所属类的一个成员函数,parameter-list 是用连字符分隔的变量或表达式的序列,它们要与该成员函数定义的参数个数及类型匹配。假设 s 是类 Shape 的实例,则用下面的语句设置线的宽度为 3:

```
s.setlinewidth(3);
```

构造方法

构造方法是一种特殊的方法。Java 中的每个类都有构造方法,用来初始化该类的一个新对象。构造方法具有和类名相同的名称,而且不返回任何数据类型。在类 Shape 中添加构造方法,并初始化线宽为 1,代码如下:



```
class Shape{
    int linewidth;
    void Shape(){
        linewidth=1;
    }
    int getlinewidth(){
        return this.linewidth;
    }
}
```

```

int setlinewidth (int width){
    this.linewidth=width;
}

```

方法重载

前面已经指出,通过方法重载可以实现多态。方法重载即指多个方法可以有相同的名字。但是这些方法的参数必须不同,或者是参数个数不同,或者是参数类型不同。下面我们重载类 Shape 的构造方法,使其接收一个整型数据作为线宽的值。



```

class Shape{
    int linewidth;
    void Shape(){
        linewidth=1;
    }
    void Shape(int width){
        linewidth=width;
    }
    int getlinewidth (){
        return this.linewidth;
    }
    int setlinewidth (int width){
        this.linewidth=width;
    }
}

```

当用运算符 new 为一个对象分配内存时,要调用对象的构造方法,而当创建一个对象时,必须用 new 为它分配内存。因此用构造方法进行初始化,避免了在生成对象后每次都要调用对象的初始化方法。如果没有实现类的构造方法,则 Java 运行时系统会自动提供默认的构造方法,它没有任何参数。

构造方法只能由 new 运算符调用。

父类、子类 and 继承

第二个基本的面向对象机制是继承。继承是关于有层次关系的类的概念。继承而得到的类为子类,被继承的类为父类,父类包括所有直接或间接被继承的类,即包括子类的所有祖先类。子类继承父类的状态和行为,同时也可以修改父类的状态或重载父类的行为,并添加新的状态和行为。一个类的后代可以继承它的祖先的所有变量和成员函数,就像创建自己的一样。一个类的直接父亲叫

做它的超类(superclass)。

在Java中,所有的类都是通过直接或间接地继承 java.lang.Object 得到的,Java中不支持多重继承。

通过在类的声明中加入 extends 子句来创建一个类的子类,其格式如下:

```
class SubClass extends SuperClass {
    .....
}
```

把 SubClass 声明为 SuperClass 的直接子类,如果 SuperClass 又是某个类的子类,则 SubClass 同时也是该类的子类。子类可以继承所有父类的内容。如果使用默认的 extends 子句,则该类为 java.lang.Object 的子类。子类可以继承父类中访问权限设定为 public、protected、friendly 的成员变量和方法,但是不能继承访问权限为 private 的成员变量和方法。下面的代码创建了一个 Shape 类,并创建它的子类 Circle:



```
class Shape{
    int linewidth;
    void Shape(){
        linewidth=1;
    }
    void Shape(int width){
        linewidth=width;
    }
    void Draw(){
        .....
    }
}

class Circle extends Shape {
    int radius;
    int x,y;
    Circle(int x,int,y,int radius){
        .....
    }
}
```

在子类中,增加了3个成员变量: x、y 和 radius, 分别表示圆的圆心坐标和半径。

覆盖

在前面的例子中,子类 Circle 的画图函数 draw()的实现和父类的画图函数

draw()的实现方法有可能不一样,可以用覆盖类解决这个问题。代码如下:



```
class Shape{
    int linewidth;
    void Shape(){
        linewidth=1;
    }
    void Shape(int width){
        linewidth=width;
    }
    void Draw(){
        .....
    }
}

class Circle extends Shape {
    int linewidth;
    int radius;
    int x,y;
    Circle(int x,int,y,int radius){
    }
    void Draw(){
        System.out.println("Circle draw()");
        .....
    }
}
```

该例中,Circle 是 Shape 的一个子类。其中声明了一个和父类 Shape 中同名的变量 linewidth,并定义了与之相同的方法 Draw,这时在子类 Circle 中,父类的成员变量 linewidth 被隐藏,父类的方法 draw 被重载。于是子类对象所使用的变量 linewidth 为子类中定义的 linewidth,子类对象调用的方法 Draw()为子类中所实现的方法。

注意,覆盖的方法和父类中被覆盖的方法要具有相同的名字,相同的参数表和相同的返回类型。子类通过成员变量的隐藏和方法的覆盖,可以把父类的状态和行为改变为自身的状态和行为。

this 和 super

Java 有一个特殊的实例值叫 this,它用来在一个成员函数内部指向当前的对象。与 this 类似,super 用来引用当前对象的父类。

`super` 的使用有 3 种情况:

- ◆ 用来访问父类被隐藏的成员变量,如:

```
super.variable
```

- ◆ 用来调用父类中被重载的方法,如:

```
super.Method([paramlist])
```

- ◆ 用来调用父类的构造函数,如:

```
super([paramlist])
```

4.2 接口

在 4.1 节中我们讨论了抽象类和抽象方法,下面介绍一个更为“抽象”的概念——接口。接口是用来说明常量和抽象方法的一种类型。接口中的方法全都是抽象方法。接口提供了方法原型的封装机制,接口和抽象类不同,它的方法不一定要限制在继承树中的某个类中实现。Java 的接口是为相互没有关系的类实现同样的一组方法而提供的一种有效的手段,也是用来弥补 Java 不支持多继承的一种方法。它的用处主要体现在下面几个方面:

- ◆ 通过接口可以实现不相关的类具有相同行为,而不需要考虑这些类之间的层次关系。
- ◆ 通过接口可以指明多个类需要实现的方法。
- ◆ 通过接口可以了解对象的交互界面,而不需了解对象所对应的类。

和类类型不同,接口可以是一个或多个其他接口的直接扩展,也就是说,接口是支持多继承的。

4.2.1 为什么使用接口

解决多继承

随着用 Java 进行面向对象编程的深入,我们会愈来愈明显地感到 Java 类层次结构的局限——它只支持单继承,一个类只能有一个超类,而不能交叉继承树的其他分支中有用的部分。当一个方法在不同的类中要有不同的实施办法,并且方法的实施办法要在运行时决定,就会出问题。这无疑与面向对象的重用思想背道而驰。

在 Java 中,定义方法的类必须在编译时出现。编译器检查方法的所属关系,看看方法调用是否合法。可以调用这个方法类都必须有共同的上级类,所以方法可以在上级类中定义并在各个子类中覆盖掉上级类中同名的方法。如果一定要使各个子类有不同的实施办法,可以把方法定义为抽象方法。我们可以把方法定义到更高的继承层次中,使更多的类能够覆盖同一方法。

怎样在类中加进一些新的特性以使这些方法能在本来无关系的类中实现? Java 的接口应运而生,利用接口 Java 就解决了多继承的问题。

概念清晰

有的语言是支持多继承的,比如 C++ 语言,但是多继承的引入使得继承层次结构变得混乱,更加容易出错和产生二义性。接口类不同,接口可以允许多继承,即一个接口可以有多个超接口。但是子接口只是从超接口那里继承了方法的声明,而不包括方法的实现和实例变量的声明。这样,就降低了完全多继承所带来的复杂性。接口和类一样,都是在源文件中声明的,建议一个文件只包含一个接口声明。同样,接口类型编译后产生的字节码文件也是以 .class 为后缀的。接口扩展了类的功能,除此以外它们几乎完全相同。但是接口不能创建实例,这是它与类不同的地方。

设计和实现的分离

除了弥补缺少多继承的缺陷外,接口还带来了新的用途。使用接口可以做到设计和实现的分离。在单继承的继承树中,设计和实现不可避免地要纠缠在一起。在设计的时候,也许只想提供一个类的抽象接口,而不希望去具体实现它,因为那是实现阶段的事情。

可以用前面说过的创建一个抽象类来完成这项工作,如果任何一个新类想使用这个接口,那么它必须是此抽象类的子类,这样一来,这个新类在继承树中的位置就固定了,它只能在此抽象类的下面。然而,如果这个新类需要作为继承树中其他某个类的子类呢? 当一个类希望同时使用多个接口又怎么办呢? 这时,抽象类就遇到了困难。

使用接口类型可以很完美地解决这个问题。一个类可以实现多个接口,而不影响它在单继承树中的位置,它还可以作为继承树中任何一个类的子类。一个类声明实现一个接口,就是承诺它将实现接口中声明的所有方法。但是抽象类可以享有一些特权,它可以只实现接口中的部分方法,但是它的任何一个非抽象子类都必须实现继承而来的所有未实现的方法。

由于接口类型有一个单独的层次结构,它们可以被混合到单继承树的类中去,这就使得您可以将一个接口插到单继承树的任何地方去。这样,单继承树就

可以仅仅被视为实现的层次,而设计的层次在多继承的接口树中。这是进行程序设计的一种非常有用的思考方法,也是一种被广泛推荐的程序设计方法,尽管开始你会不太习惯。

机制灵活

与C++不同,Java不支持多重继承,而是用接口实现比多重继承更强的功能。

多重继承使得类的层次关系不清楚,而且当多个父类同时拥有相同的成员变量和方法时,子类的行为不容易确定,这给编程带来了困难。单一继承则清楚地表明了类的层次关系,指明子类和父类各自的行为。接口则把方法的定义和类的层次区分开来,通过它可以在运行时动态地定位所调用的方法。同时接口中可以实现“多重继承”,且一个类可以实现多个接口。正是这些机制使得接口提供了比多重继承更简单、更灵活、而且更强劲的功能。

4.2.2 接口的定义

定义接口和定义类相似,只是把关键字 `class` 换成 `interface`,它的定义由两部分组成:接口说明和接口体。接口定义形式如下:

```
interface Declaration {  
    interface Body  
}
```

`Declaration` 为接口名,接口名是标识符,它不能和包中的其他类或接口重名,否则会产生编译错误。也不能和使用 `import` 语句引入的类重名。例如,在一个编译单元中出现如下的接口声明就会报错:

```
import package1.MyPoint;  
interface MyPoint  
{  
    .....  
}
```

接口声明

最简单的接口声明如下:

```
interface interfaceName{
```

```

.....
|

```

通常接口以 `able` 或 `ible` 结尾,表明接口能完成一定的行为。接口声明中还可以包括对接口访问权限以及它的父接口列表。完整的接口声明如下:

```

[public] interface interfaceName [extends listOfSuperInterface]{
.....
|

```

`interface` 前的修饰符是可选的。当没有修饰符的时候,表示此接口的访问权限是包,即只有声明它的包中的类和接口可以访问它。如果使用修饰符,则只能用 `public` 修饰符,表示此接口是公有的,在任何地方都可以引用它,这一点和类是相同的。

接口声明中的 `extends` 子句和类声明中的 `extends` 子句一样,用来定义直接超接口。当接口说明中带有 `extends` 子句时,称此接口扩展了 `extends` 子句中列出的接口,而 `extends` 后面列出的接口则是此接口的直接超接口。和类不同,一个接口可以扩展多个超接口,当 `extends` 后面有多个超接口时,它们之间用逗号隔开,比如:

```

public interface Cookable extends Foodable, Printable
{
.....
|

```

在 `extends` 子句中的每个接口类型都必须是可访问的接口类型,否则将出现编译错误。接口不允许直接或间接地扩展它自身,否则也将导致编译错误。



注意 在 Java 以前的版本中,接口前面还可以有 `abstract` 修饰符,表示此接口是抽象的。但是现在不用再使用 `abstract` 来声明,因为接口默认为抽象的。

在类中,任何 Java 的类都是 `Object` 类的子类;在接口中,不存在这样的始祖类。

接口体

接口体中声明接口的成员,包括常量和抽象方法。常量定义的格式为:

```
type NAME = value;
```

其中 `type` 可以是任意类型, `NAME` 是常量名,通常用大写, `value` 是常量值。

在接口中定义的常量可以被实现该接口的多个类共享,它与C中用`#define`以及C++中用`const`定义的常量是相同的。在接口中定义的常量具有`public`、`final`和`static`的属性。在接口的常量声明中不能使用`synchronized`、`transient`、`volatile`修饰符,否则将产生编译错误。

方法定义的格式为:

```
returnType methodName ([paramlist]);
```

接口中只进行方法的声明,而不提供方法的实现,所以,方法定义没有方法体,且用分号“;”结尾。在接口中声明的方法具有`public`和`abstract`属性。

另外,如果在子接口中定义了和父接口同名的常量或相同的方法,则父接口中的常量被隐藏,方法被重载,这一点与类一样。

一个接口中成员包括从其直接超接口中继承而来的成员,以及自己声明的新成员。从超接口中继承而来的成员包括所有超接口中的成员,除了那些被此接口隐藏的域和覆盖的抽象方法。

这里还用画图的例子。我们把`Shape`定义为接口,线的宽度为`final`型,三个方法为画图`draw()`,得到线的宽度`getlinewidth()`,设置线的宽度`setlinewidth(int newwidth)`。代码如下:



```
interface Shapeable{
    final int linewidth = 1;
    void draw();
    int getlinewidth();
    void setlinewidth(int newwidth);
}
```

4.2.3 接口的实现

接口自己不能提供方法的实现,接口中的方法必须由类来实现。类在`implements`语句中声明其实现的所有接口。`implements`语句中由关键字`implements`和多个由逗号分隔的接口组成,该语句必须放在`extends`语句之后(如果有)。

类可以实现多个接口。实现接口的类要覆盖接口及所有上级接口中定义的所有方法。但是,如果此类是抽象类,则可以把实现超接口中方法的任务交给其子类去完成,即抽象类可以只实现其超接口中的部分方法。

下面的代码是在类`Circle`中实现上面所定义的接口`Shapeable`:



```
class Circle implements Shapeable {
    void draw() {
        System.out.println("Circle implements draw()");
    }

    int getlinewidth() {
        return this.linewidth;
    }

    void setlinewidth(int newwidth) {
        this.linewidth = newwidth;
    }
}
```

在类中实现接口所定义的方法时,方法的声明必须与接口中所定义的完全一致。在 `implements` 子句中,一个接口不能出现多次,即使它们使用了不同的标识。如下面一段代码所示:



```
class Redundant implements java.lang.Cloneable, Cloneable {
    int count;
    .....
}
```

这段代码将产生编译错误,因为 `java.lang.Cloneable` 和 `Cloneable` 代表的是同一个接口,虽然所用的标识不同。

但是允许类中的一个方法实现多个超接口中的方法,比如下面的例子就可以编译成功:



```
interface Fish {
    int getNumberOfScales();
}

interface Piano {
    int getNumberOfScales();
}

class Tune implements Fish, Piano {
    int getNumberOfScales();
}
```

在此例中,类 `Tune` 中的 `getNumberOfScales()` 方法的签名与返回值和它所实现的两个接口 `Fish`、`Piano` 中的抽象方法完全相同,这时,就认为此方法实现了这两个接口中的方法。注意,在类 `Fish` 和类 `Piano` 中的两个签名相同的方法其返回值也一定要相同,否则,就无法在类 `Tune` 中用一个方法同时满足两种返回值

类型的匹配要求。

接口可以作为一种引用类型来使用。任何实现该接口的类的实例都可以存储在该接口类型的变量中,通过这些变量可以访问类所实现的接口中的方法。Java运行时系统动态地确定该使用哪个类中的方法。把接口作为一种数据类型可以不需要了解对象所对应的具体的类,而着重于它的交互界面,如前面所定义的 Shapeable 接口和实现该接口的 Circle 类。下例中,我们以 Shapeable 作为引用类型来使用。



```
class testInterface{
    public static void main( String args[] ){
        Shapeable s = new Circle();
        s.draw();
    }
}
```

4.3 包

包是 Java 用来设计较大规模的程序而引入的概念。包用来分类组织众多的类,它是相关类和接口的集合。由于 Java 编译器为每个类生成一个字节码文件,且文件名与类名相同,因此同名的类有可能发生冲突。为了解决这一问题,Java 提供包来管理类名空间。包实际上提供了一种命名机制和可见性限制机制。

4.3.1 为什么要使用包

在以前举的一些例子程序中,基本上都没有使用包,因为例子通常都是一些比较小的示例性的程序,只要两三个类就可以完成了,不需要再进行分类组织。但是,当开发较大规模的程序时,就会发现目前这种开发模式的局限。

当创建了大量的类以后,会对大量冗长的名字感到厌烦,于是可能会趋向于使用较短小的名字。使用简短的名字固然会给编程带来便利,但是当类的数量激增时,这些类就有可能重名,而也许还不知道。使用包就可以解决这个问题,可以使用包将类组织起来,而在包内可以使用简单名称,在越包访问的时候就需要使用带包名的限定名称。

使用包不仅可以避免名字的冲突,还可以将大量的类按照功能和用途组织起来,访问和查找起来更加方便。

可以把包理解为一种组合机制,包主要有 2 种用途:

- ◆ 减少命名混乱问题。
- ◆ 控制包中类、接口、方法和一定的访问权限控制。

使用包可以提供一定的访问控制。类、方法和域的默认访问权限就是可以在声明它的包中被访问。包中封装了类和其他的包,所以 Java 提供了对类成员在 4 种范围中的访问权限的控制,这 4 种范围包括:

- ◆ 同一个类中。
- ◆ 同一个包中。
- ◆ 不同包中的子类。
- ◆ 不同包中的非子类。

访问权限则包括 `private`, `protected`, `public` 和 `friendly`。表 4.2 列出了在不同范围中的访问权限。

表 4.2 访问权限控制

	同一个类中	同一个包中	不同包中的子类	不同包中的非子类
<code>private</code>	✓			
<code>protected</code>	✓	✓	✓	
<code>public</code>	✓	✓	✓	✓
<code>friendly</code>	✓	✓		

“✓”表示可以访问。例如:如果一个类成员的访问权限控制为 `protected`,则处在另一个包中的它的子类可以访问该成员。

4.3.2 包与类名

类只有在属于不同的包时才可以同名。当类被程序引用时,编译器会检查所引用的包,找出类的定义语句。

如果只有一个包中定义了要使用的类,则使用该包中的定义。如果多个引入包中包含类定义语句,则必须用包名作为前缀以避免类产生歧义性。类名和包名之间用“.”作为分隔符。例如,在自定义的两维图形包 `graphic2d` 和三维图形包 `graphic3d` 中都定义一个 `Point` 类来指向用户定义的点,一个是二维点 `x,y` 坐标,一个三维是 `x,y,z` 坐标,如果两个包都被引入到程序中,则必须用 `graphic2d.Point` 和 `graphic3d.Point` 分别指向二维和三维的 `Point` 类。

包可以嵌套形成正确的层次。这种嵌套能力可以帮助我们进一步组合类,

就像目录可以帮助组织文件一样。为了引用一个嵌套包中的类,可以使类名含有所有包名前缀。例如,为了引用 java 包中 awt 包的类,完整的类名是 java.awt.Point。如果引入 java.awt 并且该类不会与其他引入包的类名相混,则可以直接用 Point 引用这个类。

4.3.3 包与目录

每个包应映像到文件系统中同名的子目录中。包的嵌套反映了文件系统中的目录层次。在 Java 中,类是按照包分类组织的,包采用类似目录的层次结构,在一个目录中包括文件和子目录。

类似的,在一个包中包括:类文件、接口文件(一个类或一个接口编译后都产生一个相应的字节码文件)和子包。

由于包的层次结构类似目录的层次结构,所以,为了便于实现,Java 对包的存储作了一些规定。使用文件系统来存放包时,系统中所有的 Java 包、类的源文件、字节码文件都必须存放在单一的目录和它的子目录中。Visual J++ 6.0 中有对象浏览器,可以查看各个包中的对象。如图 4.1 所示:

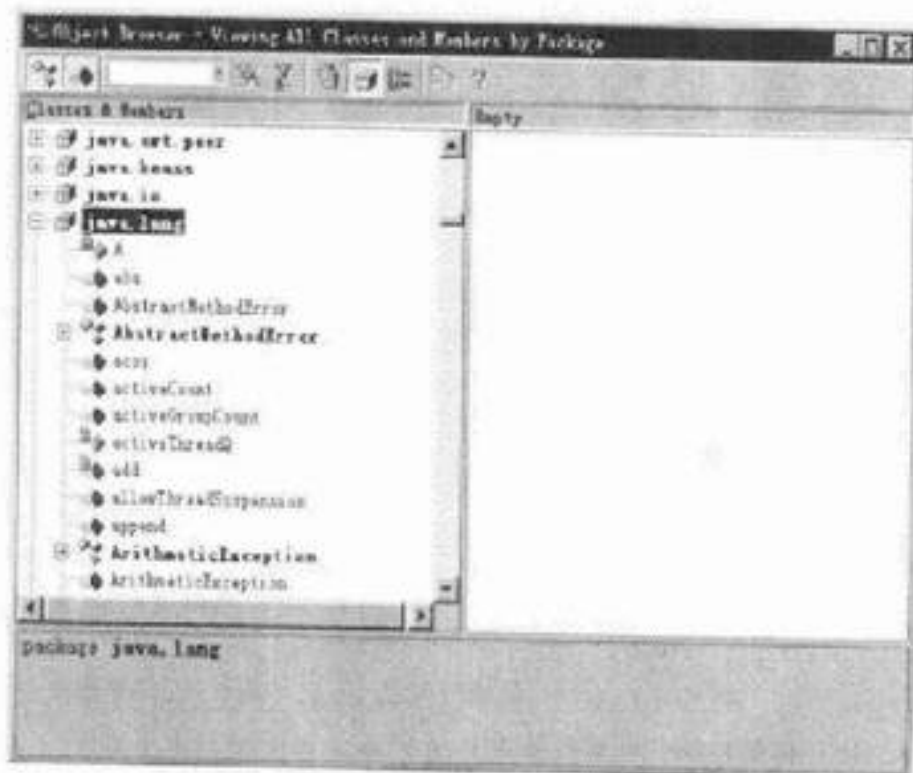


图 4.1 对象浏览器

4.3.4 包(package)语句

package 语句作为 Java 源文件的第一条语句,指明该文件中定义的类所在的包。它的格式为:

```
package pkg1[.pkg2[.pkg3...]];
```

Java 编译器把包对应于文件系统的目录管理。例如:名为 myPackage 的包中,所有类文件都存储在目录 myPackage 下。同时,package 语句中,用“.”来指明目录的层次,例如:

```
package java.awt.image;
```

指定这个包中的文件存储在目录 java/awt/image 下。一个程序文件只能有一个包语句,且必须作为文件中的第一条非说明语句。如果程序文件中省略包语句,则生成的类,在引用时使用非命名的默认包。

所有 Java 内置类(即 Java 核心 API)都放在 Java 包中。Java 包中定义了 12 个子包,如表 4.3 所示。

表 4.3 java 包中的子包

包	说明
java.applet	小程序,包括一个语音片断接口
java.awt	图形用户接口组件和绘图功能
java.beans	JavaBeans 支持类
java.io	输入输出流和文件
java.lang	基本类,如字符串、对象和线程
java.math	扩展精度运算的类
java.net	与网络相关的类
java.rmi	远程方法调用类
java.security	安全操作相关的类
java.sql	对关系数据发送 SQL 语句的类
java.text	对国际化文本处理的相关类
java.util	用于特殊数据结构的实用程序类

在 Visual J++ 中,包被打包成 .zip 文件的形式,如果想了解 Java 包的目录结构和类文件,可解压系统目录中\Java\Package 目录下的某个 zip(如,135JZDNZ.ZIP)文件到一临时目录,结果如图 4.2 所示。



图 4.2 java 包的存储目录结构

4.3.5 import 语句

前面讲述了如何定义一个包,从某种意义上讲,使用包更为重要。因为 Java 标准类库以及其他公司(如 Microsoft)都是以包的形式提供的。实际上,在前面所举的示例程序中,当我们使用 Java 标准类库中的类时,我们已经用到了包,只是没有显式地说明。为了使用 Java 中已提供的类,需要用 import 语句来引入所需要的类。下面是两种 import 语句:

```
import packageName.ClassName
import packageName.
```

其中 packageName 是单个包名或多个用“.”号隔开的嵌套包名。在第一种形式中,只引用命名包中的指定类;在第二种形式中,命名包中定义的所有类和接口都可以直接用简单名称访问。import 语句必须放在程序文件的开头。并且其前面允许的唯一非说明语句是 package 语句。

在下面的例子中,引入 java.util 包的类和 Java.awt 包中的所有类和接口,但不引入子包 java.awt.image 中定义的类和接口。因此,Vector 可以直接用简单名称,Java.awt 包中定义的类,如 label 和 button 也是如此。但是,java.io 包和包中定义的 PrintStream 类都没有引入,PrintStream 必须用全名引用。



```
import java.util.Vector;
import java.awt.*;
Class Importtest{
    Vector label;
    Button button;
    Java.io.PrintStream out
}
```

Java 编译器为所有程序自动引入包 `java.lang`, 因此不必用 `import` 语句引入它包含的所有的类, 但是若需要使用其他包中的类, 必须用 `import` 语句引入。

引入了类和包并不表示运行时要装入这个类或包中的类。`import` 语句只是给出编译器寻找类名的暗示, 引入语句中提到的类和包可能在程序体中永远用不到。



更上一层楼

在本章中, 我们学习了面向对象编程的概念, 重点学习了 Java 中的类、抽象类和抽象方法、包以及接口。对于每个概念, 我们讨论了在这些概念中有哪些新的思想, 以及这些思想是如何有利于编程的。然后详尽地讲述了这些概念的具体语法, 并结合实例进行分析。接口则在抽象类的基础上更进了一步, 同时它还支持多继承, 从而弥补了 Java 单继承的缺陷。Java 是一种面向对象的语言, 应该说我们对 Java 面向对象的讨论是不够的。限于篇幅, 对一些概念, 如抽象类、抽象方法、`final` 等, 没有进行深入的讨论。我们的例子非常简单, 它们有助于理解这些概念, 要完全理解 Java 面向对象的思想, 这些例子是远远不够的。第 3 部分我们将用 Visual J++ 进行 Java 编程, 来帮助读者更深入地理解 Java 面向对象的思想。

第5章 Java 应用程序分类

知识要点:

- ◆ Java 应用程序分类
- ◆ Java 小程序和应用程序的差别
- ◆ Java 小程序的特点
- ◆ Java 应用程序的特点
- ◆ 如何利用 Visual J++ 创建小程序
- ◆ 如何利用 Visual J++ 创建应用程序

在前面的4章中,我们了解了 Visual J++ 的开发环境和 Java 语言的概况,包括 Java 编程基础、面向对象的特性、类、接口和包等。从本章开始,我们利用 Visual J++ 开发环境,进行 Java 编程。本章我们系统介绍 Java 应用程序分类,小程序和应用程序的区别。利用 Visual J++ 开发环境,通过两个例子,简单了解小程序和应用程序的特点。



光盘 参阅本书配套光盘的【创建小程序】部分可交互学习与本章相关的知识。

5.1 小程序和应用程序比较

5.1.1 Java 应用程序分类

Java 程序有 3 种类型：小程序、小服务程序和应用程序。简单地说，小程序就是嵌入 Web 文档的程序；小服务程序是在 Web 服务器内运行的特殊程序；而 Java 应用程序是所有其他类型的程序，如网络服务器和家用电器上使用的程序。

小程序实际上是编译后的 Java 类，它运行在 Web 浏览器的网页中。由于 Web 浏览器加载小程序并提供小程序的主窗口，小程序与应用程序比较简单。Java 应用程序不需要附加的软件，可独立运行。

5.1.2 小程序和应用程序的比较

在 Java 中，小程序是从 Web 网页中加载的 Java 程序，Java 应用程序则是领航系统上运行的程序。对 Java 而言，小程序的大小和复杂性都没有限制。事实上，小程序有的方面比 Java 应用程序功能强大。由于网络速度的限制，Java 小程序规模很小。

Java 小程序和 Java 应用程序共享许多资源和特征。小程序和应用程序之间的技术差别来源于其运行环境的差别。Java 应用程序可运行在最简单的环境中，它的唯一输入就是来自外部的命令行参数。另一方面，Java 小程序则需要来自 Web 浏览器的大量信息：它需要知道何时启动，何时放在浏览器窗口，何处、何时激活或关闭。由于执行环境的不同，Java 小程序和应用程序的最低要求不同。

作为网络编程语言，Java 小程序可以嵌入到 Web 网页中，在制作动态网页、交互性强的网页方面有着不可替代的作用；另一方面，由于 Java 应用程序不能嵌入到 Web 网页中，在解决传统的客户/服务器结构的问题，尤其是在跨平台的图形界面应用程序方面，和其他语言相比具有绝对优势。

小程序和应用程序的比较见图 5.1。

从 Java 数据结构来讲，小程序和应用程序的区别在于：小程序是从 `java.applet.Applet` 类派生而来。由于从 `java.applet.Applet` 继承了大量的方法，小程序编程相对简单。Java 应用程序与传统的 C、Pascal 类似，主要利用 `main()` 函数来完成应用程序的功能。

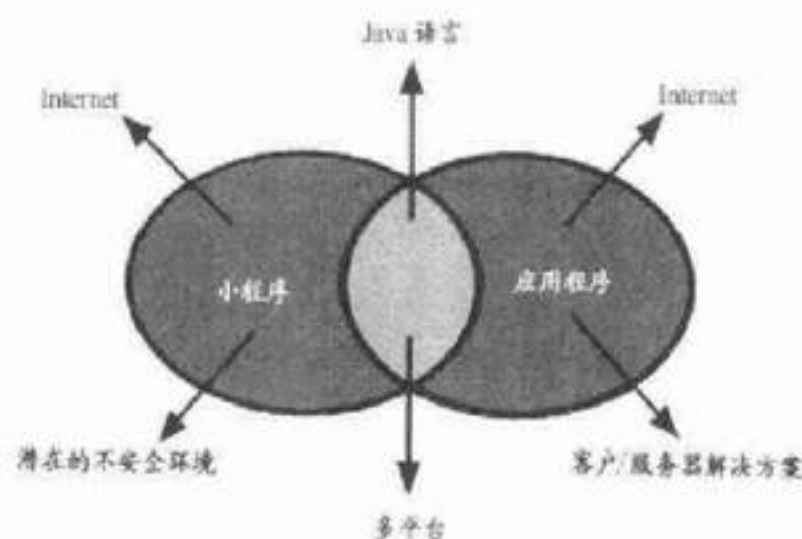


图 5.1 Java 小程序和应用程序的比较

决定编写 Java 小程序还是应用程序取决于程序的运行环境及其传递机制。由于 Java 小程序总是放在 Web 浏览器的图形用户界面环境中,当图形显示不理想时,Java 应用程序比 Java 小程序更合适。例如,用 Java 编写的 Web 服务器就不需要图形显示,只需文件和网络访问。

Web 协议使 Java 小程序发布很方便,因此 Java 小程序更适合 WWW 上的应用程序,尽管 Java 应用程序也能很方便地完成许多相同的任务。非网络系统和内存较小的系统更适合用 Java 应用程序,Java 应用程序可以不用图形界面,提供 Java 运行环境(JRE)的系统可运行 Java 应用程序,而小程序必须有 Web 浏览器的支持。表 5.1 列出了 Java 小程序和应用程序的差别。

表 5.1 Java 小程序和应用程序的差别

	Java 小程序	应用程序
图形	固定使用图形	可选
运行要求	Java 程序加 Web 浏览器	最低 Java 应用程序要求
输入	浏览器客户机的位置和大小,嵌入 Web 网页中的参数	命令行参数
虚拟机所需过程	Init, start, stop, destroy, paint	Main——启动过程
典型应用	动态、交互式网页	网络服务器、家电和电器控制
网络环境	一般是网络环境	可用在非网络环境

5.2 小程序

小程序是一种被嵌入到 Web 页中,由支持 Java 的浏览器解释执行的程序。使用 Java 小程序,可以大大提高 Web 页的交互能力和动态执行能力。嵌入了 Applet 的 Web 页被称为有 Java 能力的 Web 页,而可以运行小程序的浏览器被称为 Java 兼容的浏览器。目前,市场上占主流的浏览器——IE 和 Navigator,都是 Java 兼容的浏览器。不支持 Java 的浏览器也可以浏览有 Java 能力的 Web 页,但是其中的小程序将被忽略。

5.2.1 Applet 类的继承关系

在 Java 中,每个小程序都是由 Applet 类的子类实现的,Applet 类在 Java 类库的 `java.applet` 包中定义,图 5.2 说明了 Applet 类的继承关系。

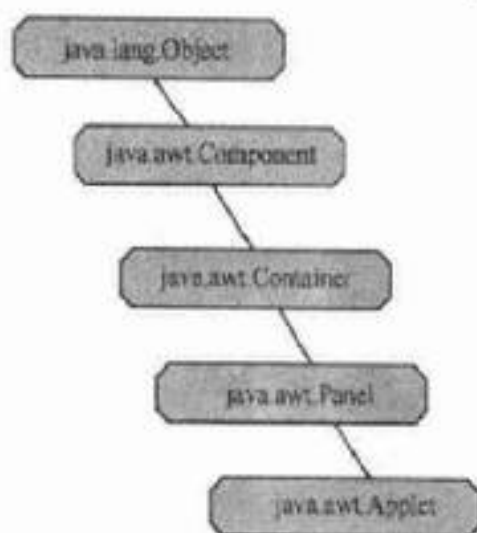


图 5.2 Applet 类的继承关系

有关 applet 类的详细讨论见第 6 章,下面我们用一个例子说明如何创建 Java 小程序。

5.2.2 创建 Java 小程序

► 在 Visual J++ 中创建 Java 小程序

1. 运行 Visual J++, 打开 File 菜单,选择 New Project 命令,弹出 New Project 窗口。

2. 在窗口左边的一栏中打开 Visual J++ Projects 文件夹, 选择 Web Pages 命令。
3. 然后在右边的一栏中选择 Applet on Html 文件, 在下面的 Name 栏中可以修改工程名, 这里我们键入“ex05a”, 然后单击【打开】按钮, 如图 5.3 所示。

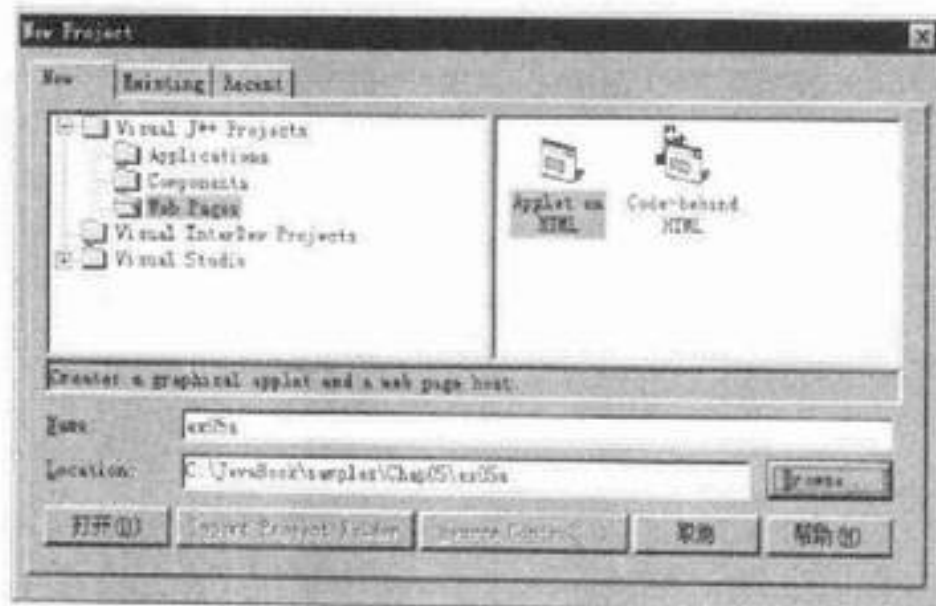


图 5.3 New Project 窗口

4. Visual J++ 会生成 Java 小程序的框架文件和代码。在 Visual J++ 开发环境下, 单击工程资源浏览器中的 ex05 工程, 工程 ex05 下有两个文件, Applet1.java 和 Page1.htm, Applet1.java 为我们创建 Java 小程序的源代码, Page1.htm 为嵌入此小程序的 Web 网页。

Applet1.java 的源代码如下:



```
import java.awt.*;
import java.applet.*;

/**
 * This class reads PARAM tags from its HTML host page and sets
 * the color and label properties of the applet. Program execution
 * begins with the init() method.
 */
public class Applet1 extends Applet
{
    /**
     * The entry point for the applet.
     */
}
```

```
public void init()
{
    initForm();

    usePageParams();

    // TODO: Add any constructor code after initForm call.
}

private final String labelParam = "label";
private final String backgroundParam = "background";
private final String foregroundParam = "foreground";

/*
 * Reads parameters from the applet's HTML host and sets applet
 * properties.
 */
private void usePageParams()
{
    final String defaultLabel = "Default label";
    final String defaultBackground = "COCOCO";
    final String defaultForeground = "000000";
    String labelValue;
    String backgroundValue;
    String foregroundValue;

    /*
     * Read the <PARAM NAME="label" VALUE="some string">,
     * <PARAM NAME="background" VALUE="rrggbb">,
     * and <PARAM NAME="foreground" VALUE="rrggbb"> tags from
     * the applet's HTML host.
     */
    labelValue = getParameter(labelParam);
    backgroundValue = getParameter(backgroundParam);
    foregroundValue = getParameter(foregroundParam);

    if ((labelValue == null) || (backgroundValue == null) ||
        (foregroundValue == null))
    {
    }
}
```

```

    /**
     * There was something wrong with the HTML host tags.
     * Generate default values.
     */
    labelValue = defaultLabel;
    backgroundValue = defaultBackground;
    foregroundValue = defaultForeground;
}

/**
 * Set the applet's string label, background color, and
 * foreground colors.
 */
label1.setText(labelValue);
label1.setBackground(stringToColor(backgroundValue));
label1.setForeground(stringToColor(foregroundValue));
this.setBackground(stringToColor(backgroundValue));
this.setForeground(stringToColor(foregroundValue));
}

/**
 * Converts a string formatted as "rrggbb" to an awt.Color object
 */
private Color stringToColor(String paramValue)
{
    int red;
    int green;
    int blue;

    red = (Integer.decode("0x" + paramValue.substring(0,2))).
        intValue();
    green = (Integer.decode("0x" + paramValue.substring(2,
        4))).intValue();
    blue = (Integer.decode("0x" + paramValue.substring(4,
        6))).intValue();

    return new Color(red,green,blue);
}

/**
 * External interface used by design tools to show properties of
 * an applet.
 */
public String[][] getParameterInfo()
{

```



```
◆ private void usePageParams()
```

`usePageParams()`为私有方法,用来得到相关网页中<PARAM>标记的参数,设置小程序的前景色和背景色。如果`usePageParams()`不能得到这些参数值,就使用默认值。

```
◆ private Color stringToColor(String paramValue)
```

stringToColor(String paramValue)方法把 paramValue 字符串转化为颜色值,并返回该值。

```
◆ public String[][] getParameterInfo()
```

`getParameterInfo()`是重载 `applet` 类中的方法,返回小程序可理解的参数信息。Java 小程序应重载此方法,以返回正确的信息,如果此方法不被重载,将调用父类 `applet` 类中的此方法,返回 `NULL` 值。

◆ void initForm()

initForm()初始化小程序的前景色和背景色,在小程序中添加 java.awt.Label 组件。

Page1.htm 源代码如下:



```
< HTML >
< HEAD >
< META NAME = "GENERATOR" Content = "Microsoft Visual Studio 6.0" >
< /HEAD >
< BODY >

< P > &nbsp; < /P >

< ! --Insert HTML here -- >
    < applet
        code = Applet1.class
        name = Applet1
        width = 320
        height = 200 >
        < param name = label value = "This string was passed from the
HTML host." >
        < param name = background value = "008080" >
        < param name = foreground value = "FFFFFF" >

< /BODY >
< /HTML >
```

Page1.htm 用于测试 Java 小程序, < applet> 和 </applet> 中的代码为嵌入小程序代码, 并设置了小程序的参数。

5.2.3 运行结果

按 F5 键(或者打开 Debug 菜单, 单击 Start 命令)运行工程, 结果如图 5.4 所示:

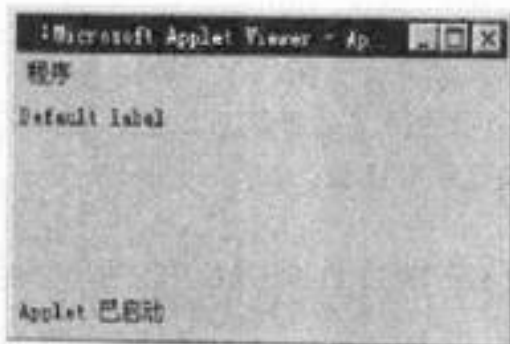


图 5.4 ex05a 运行结果

在浏览器中打开 Page1.htm, 结果如图 5.5 所示:



图 5.5 ex05a 在 IE 4.0 中运行结果

5.3 应用程序

Java 应用程序不同于 Java 小程序,不用派生 Java 类库生成 Java 应用程序,没有可重载的方法来执行。Java 应用程序由含有 `main()` 方法的类组成,`main()` 方法与 C/C++ 类似,Java 应用程序是从 `main()` 方法开始执行的。

5.3.1 创建应用程序

Visual J++ 提供了 3 种 Java 应用程序: Windows Application、Console Application 和 Application Wizard。

► 用 Application Wizard 创建应用程序

1. 在 Visual J++ 环境下,打开 File 菜单,选择 New Project 命令,弹出 New Project 窗口。
2. 在窗口左边的一栏中打开 Visual J++ Projects 文件夹,选择 Applications 命令。
3. 然后在右边的一栏中选择 Application Wizard 文件。
4. 在下面的 Name 栏中可以修改工程名,这里我们键入“ex05b”,然后单击【打开】按钮,如图 5.6 所示。

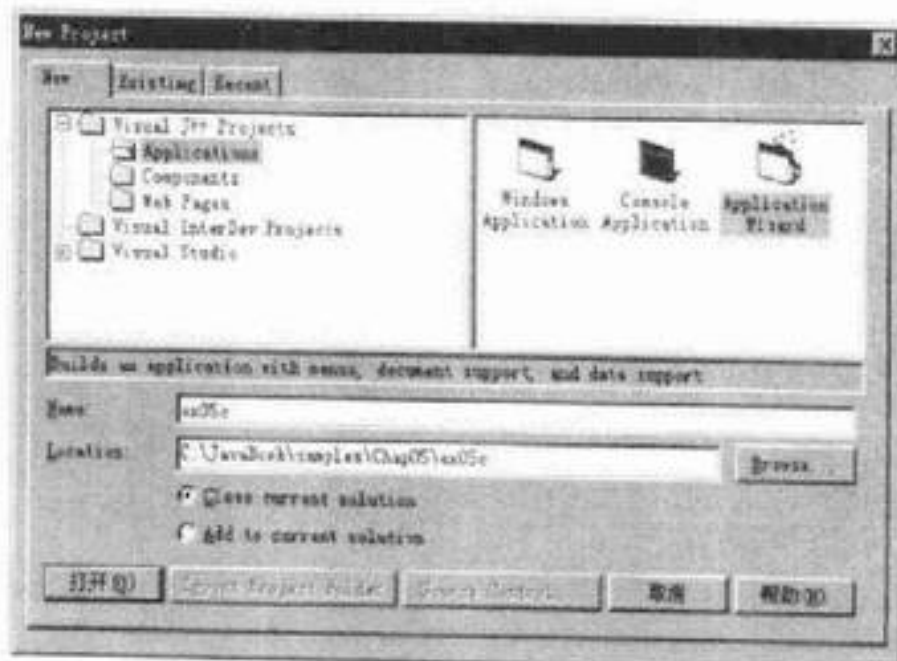


图 5.6 New Project 窗口

5. 由于利用了向导方式,接下来会有一系列窗口询问应用程序的特点,为简单起见,我们都选择默认选项,直接单击 Next 按钮即可。此向导共 6 步,最后 1 步如图 5.7 所示。

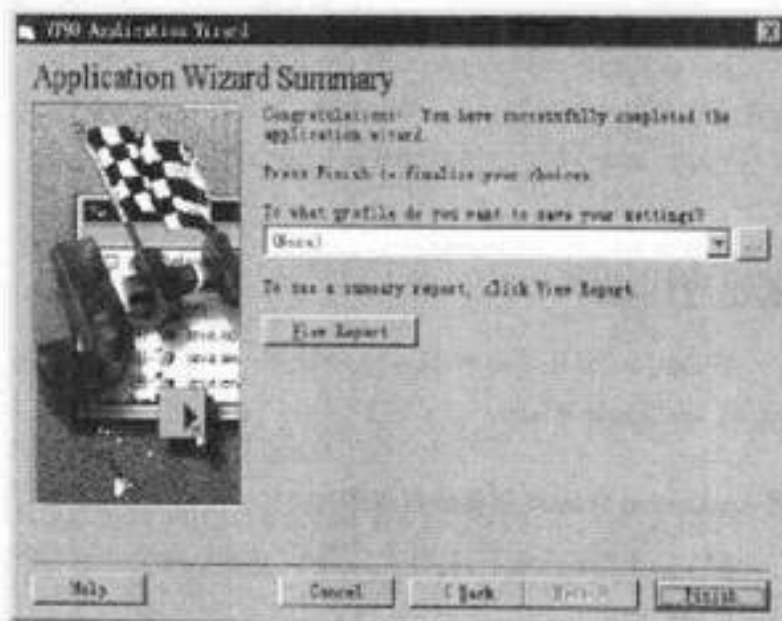


图 5.7 应用程序向导第 6 步

6. 单击 Finish 按钮, Visual J++ 生成应用程序框架。

单击工程资源浏览器中的 ex05b 工程,工程 ex05b 下有 3 个文件,About.java、ex05b.java 和 ex05b.resources。应用程序的主要文件为 ex05b.java。在 ex05b.java 的源代码中,有关 Menu、Edit、Tool bar、Status bar 的方法在第 8 章讨论,ex05b 类中有 main() 方法,代码如下:



```
public static void main(String args[])
{
    Application.run(new ex05b());
}
```

Application 类为 com.ms.wfc.app 包中的应用程序类,它的声明如下:

```
public final class Application
```

在 ex05b.java 中,main 方法调用 Application 类的静态方法 run 来创建主窗体。run() 方法的声明如下:

```
public static void run()
public static void run( Form mainForm )
```

在 `main()` 方法中,调用 `Application.run()` 方法时,传递一个 `ex05b` 的实例作为参数。而 `ex05b` 类为 `Form` 类的子类,即为应用程序的主窗体,同时当前线程开始相关的消息循环。

5.3.2 运行结果

按 `F5` 键(或者打开 `Debug` 菜单,单击 `Start` 命令)运行工程,结果如图 5.8 所示。

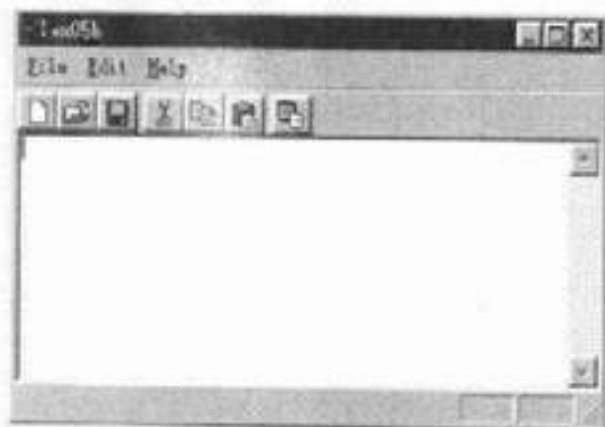


图 5.8 `ex05b` 运行结果

这是 `Visual J++ Application Wizard` 为我们生成的标准应用程序,我们没有编写一行代码,它的功能也很简单,只有编辑、复制、粘贴功能。第 8 章、第 9 章我们将介绍一些复杂的例子。



更上一层楼

本章我们系统介绍了 Java 应用程序分类,小程序和应用程序的区别。开发小程序和应用程序可达到事半功倍的效果,我们举的两个例子都非常简单,没有编写一行代码,目的是通过例子来说明小程序和应用程序的区别及利用 `Visual J++` 如何快速开发 Java 小程序和应用程序。有关 Java 小程序和应用程序更深层次的内容,我们将在后面的章节中讨论。读者可以尝试着自己添加一些代码,如显示一些文字或显示时间等。

第6章 Java 小程序编程入门

知识要点:

- ◆ 用 Visual J++ 创建 Java 小程序
- ◆ Applet 类和 AWT 包的含义
- ◆ Java 小程序如何对事件进行处理
- ◆ 如何把 Java 小程序嵌入到 Web 页
- ◆ 如何使用 Applet 参数

在本章中,我们将创建一个 Java 小程序,对如何创建 Java 小程序进行全面讲解,并对 Java 小程序的一些重要概念以及 Visual J++ 的一些主要特性进行介绍,最后介绍几个 Java 小程序的精彩实例。



光盘 参阅本书配套光盘的【创建小程序】部分可交互学习与本章相关的知识。

6.1 一个简单的例子

我们从一个简单的例子开始,这个例子将不利用 Visual J++ 小程序模板,而是创建一个新的空工程,从头创建小程序。

6.1.1 新建工程

► 新建工程

1. 从 File 菜单中选择 New Project 命令,显示 New Project 窗口(如图 6.1 所示)。

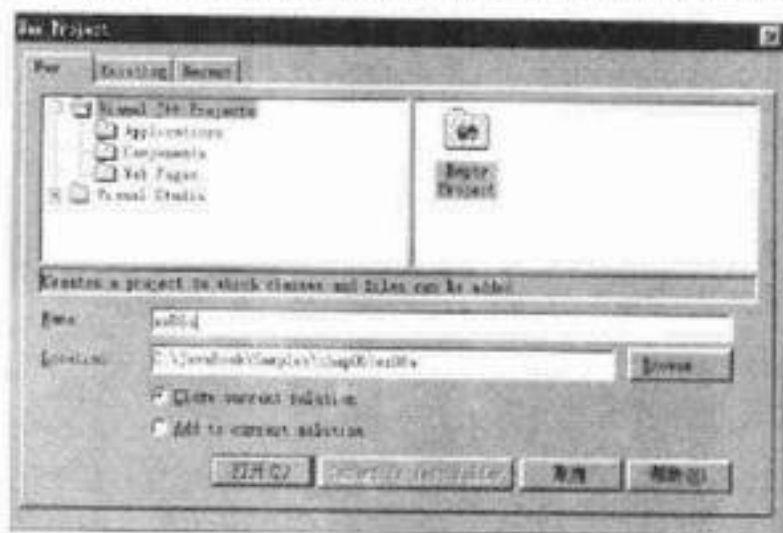


图 6.1 New Project 窗口

2. 在 New Project 窗口左边的一栏中选择 Visual J++ Projects 文件夹。在右边的一栏中选择 Empty Project 文件夹。在下面的 Name 栏中可以修改工程名,这里我们键入“ex06a”。
3. 然后单击【打开】按钮。

6.1.2 给工程添加类

► 添加类的步骤如下:

1. 打开空工程后,在 Visual J++ 的集成开发环境右边的 Project Explorer 面板

中,右击工程图标“ex06a”,显示快捷菜单(如图 6.2 所示)。选择 Add 栏目下的 Add Class 项,给工程添加一个类。



图 6.2 在 Project Explorer 面板上右击工程图标后的快捷菜单

2. 这时显示 Add Item 窗口(如图 6.3 所示),在窗口左边栏中选定 Class,在右边栏中选择 Class,并在下面的 Name 框中以默认值 Class1 作为类名(.java 后缀表示将要新建的是 Class1 类的源文件)。单击【打开】按钮,在工程中加入 Class1 类。当然也可以在 Name 框中改变类的名字。

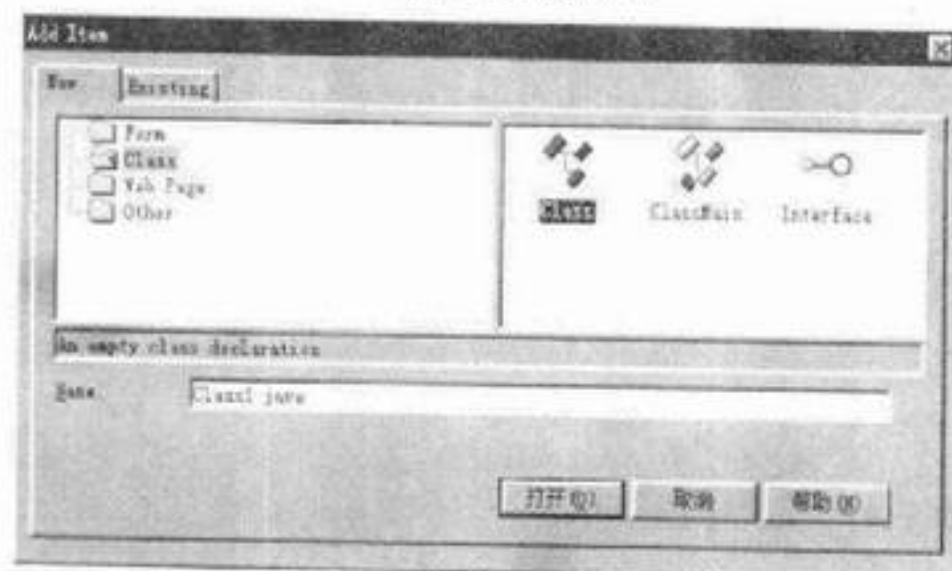


图 6.3 Add Item 窗口

3. 现在我们可以看到一个代码窗口,它位于 Visual J++ 集成开发环境的中间,

代码窗口中目前只有一个空类,显示出的 class1.java 文件的源代码如下:



```
public class Class1
```



注意 手工定义类的时候,一定注意类名的大小写,务必使程序中类的名称与相对应的文件的名称保持一致,否则会出现错误信息。这是因为 Java 是区分字母大小写的。

6.1.3 给类中引入包

现在,在“public class Class1”语句前添加下面两条 import 语句,引入 java.applet 和 java.awt 包。



```
import java.applet.*;  
import java.awt.*;
```

此举目的是为了在程序中能够调用这两个包中的类、成员变量和方法,这与 C 或 C++ 语言中的 include 语句是相似的。但是在 Java 中包是作为一种层次结构存在的,可以只装入需要的部分,这里装入了 java.applet 和 java.awt 包中所有的类、成员变量和方法。

引入 java.applet 包后,就可以修改 Class1 类的声明部分,确定 Class1 类的直接父类为 Applet,即 Class1 类由 Applet 派生而来,这样就可以把 Class1 类同定义在 java.applet 包中的 Applet 类连接起来。方法是在 Class1 类名后面键入“extends Applet”,修改后的代码如下所示:



```
import java.applet.*;  
import java.awt.*;  
  
public class Class1 extends Applet  
{  
}
```

6.1.4 给类中添加方法

► 现在来加入 java.awt 包中定义的 paint() 方法

1. 右击代码窗口中任意一处,显示快捷菜单,如图 6.4 所示。



图 6.4 代码窗口的快捷菜单

2. 选择快捷菜单上的 Sync Class Outline 命令, 然后会显示当前小程序的 Class Outline 窗口(如图 6.5 所示)。

还有另一种打开当前小程序的 Class Outline 窗口的方法: 在 View 菜单中选择 Other Windows 下的 Document Outline 命令。

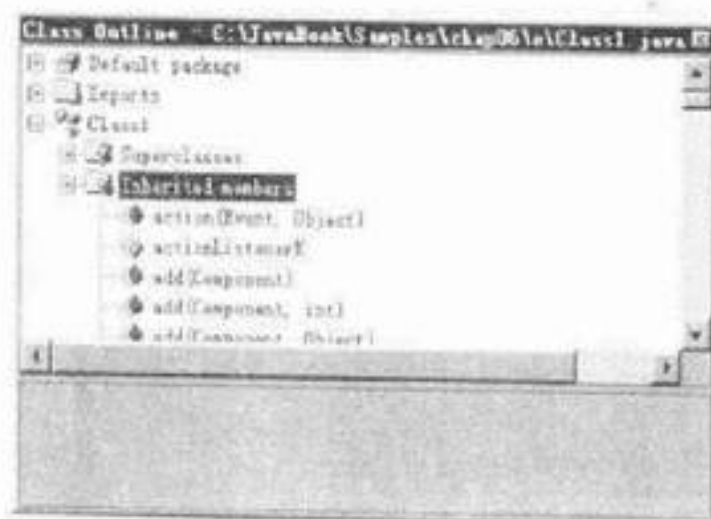


图 6.5 Class Outline 窗口

3. 打开 Class1 下的 Inherited members 项, 该项下面列出了 Class1 所继承的所有成员变量和方法, 这些成员变量和方法是按字母排序的, 按键盘上的 p 键可以跳到以 p 开头的成员变量和方法, 找到 paint(Graphics) 方法, 右键单击该方法的图标, 显示快捷菜单(如图 6.6 所示)。



图 6.6 在 Class Outline 窗口中右键单击方法图标后的快捷菜单

4. 选择 **Override Method** 命令, `paint(Graphics)` 方法就会自动添加到 `Class1` 类中, 关闭掉 `Class Outline` 窗口。

可以看到 `paint(Graphics)` 方法已经添加到程序中, 代码窗口中显示代码如下:



```
import java.applet.*;
import java.awt.*;

public class Class1 extends Applet
{
    .public void paint(Graphics p1)
    |
    // TODO: Add your own implementation.
    super.paint(p1);
}
|
```

实际上, `paint` 方法来自包 `java.awt` 中的 `Container` 类。为什么这么说呢? 我们来看一下。在上面所说的 `Class Outline` 窗口中找到 `paint(Graphics)` 方法, 用鼠标双击该方法的图标, 则会显示 `Object Browser` 窗口(如图 6.7 所示)。



图 6.7 Object Browser 窗口

Object Browser 窗口左边栏列出了所有的类,在右边栏显示的是 Container 类的所有成员变量和方法,可以看到左边栏中的 Container 类和右边栏中 Container 类的方法 `paint(Graphics)` 是自动被选中的。下面的状态栏显示出所选项 `paint(Graphics)` 方法的相关信息,“`public void paint(java.awt.Graphics g)`”是 `paint` 方法的声明部分,表明调用 `paint` 方法的方式,“Member of java.awt.Container”则说明 `paint` 方法是 Container 类的成员。

6.1.5 在方法中添加自己的语句

在 `paint(Graphics)` 方法中的 `super.paint(g)` 语句下面添加如下语句:



```
g.drawString("欢迎来到 Java 的世界!",50,50);
```

这条语句用于在屏幕上 (50,50) 坐标处显示“欢迎来到 Java 的世界!”这几个字。在本章后面的例子中我们将在这个例子的基础上增加新的功能。

从程序中 `paint` 方法的开头部分 `public void paint(Graphics g)`, 可以看出方法的参数 `g` 属于 Graphics 类。上面新添加的语句“`g.drawString("欢迎来到 Java 的世界!",50,50);`”实际上是调用了 Graphics 类的 `drawString` 方法。Graphics 类包含很多有用的绘图函数: 直线、圆、盒等等,您可以试试 Graphics 类中的其他函数。只要用不同的函数替换上面的 `paint` 方法就可以看到它们的效果。也可以试一下 Graphics 类中的 `set` 函数,例如下面的代码演示了如何使用颜色,运行后“欢迎来到 Java 的世界!”这几个字将显示为红色。



```
import java.applet.*;
import java.awt.*;

public class Class1 extends Applet
{
    public void paint(Graphics g)
    {
        // TODO: Add your own implementation.
        super.paint(g);g.setColor(Color.red);
        g.drawString("欢迎来到 Java 的世界!", 50, 50);
    }
}
```


6.1.6 运行结果

按 F5 键(或者选择 Debug 菜单下的 Start 命令)运行工程。如果是第一次运行,会出现如图 6.8 所示的窗口。这是工程 ex06a 的属性窗口,其中 Launch 属性页是要设置工程的运行方式,默认设置为调用程序 WJView.exe 来执行工程。这里,我们用默认设置,直接单击 OK 按钮。要打开工程 ex06a 的属性窗口,还可以通过 Project 菜单中的 ex06a Properties 命令。

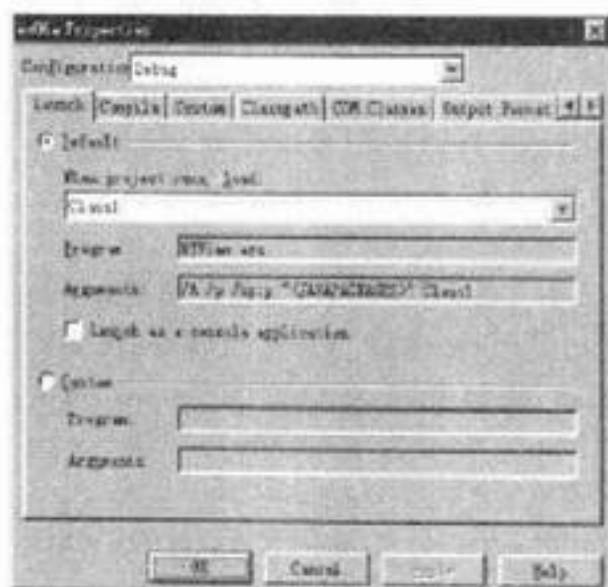


图 6.8 工程 ex06a 的属性窗口

单击 OK 按钮后,会出现 Microsoft Applet Viewer 窗口,结果如图 6.9 所示:

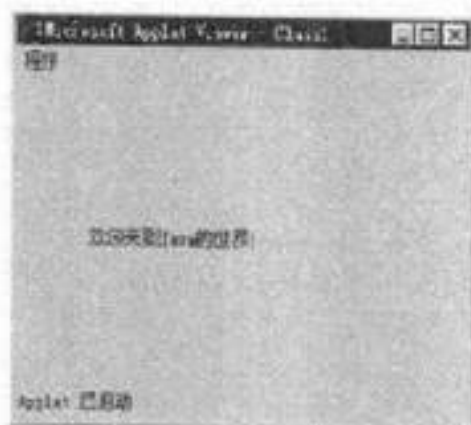


图 6.9 “欢迎来到 Java 的世界!”程序结果

实际上,程序结果的出现经过了两个过程,首先是编译,其次是运行。编译和运行 Java 小程序需要 Java 编译器和支持 Java 的浏览器,这里用到的 Java 编译器是由 Visual J++ 集成开发环境提供的内部编译器,而支持 Java 的浏览器即为程序 WJView.exe。后面我们要讲到如何把小程序嵌入到 Web 页,那时 Java 小程序将运行在支持 Java 的 Web 浏览器中。

6.2 Applet 类

6.2.1 理解程序

在前面的程序中,一开始就利用“import java.applet.*;”语句引入了 java.applet 包,并通过在 Class1 类名后面加入“extends Applet”的方法,使 Class1 类同定义在 java.applet 包中的 Applet 类连接起来,参看下面的语句。



```
public class Class1 extends Applet
{
}
```

程序中引入 java.applet 包,并使 Class1 类同 Applet 类连接起来,这本质上是由 Applet 类派生出新类——Class1 类,从而使 Class1 类继承 Applet 类中所有的成员变量和方法。这样,Class1 类就具备了 Applet 类的所有功能和特点,例如自动调用 Applet 类中定义的 init()、start()、stop()、destroy() 等方法,我们将在后面讲到这些方法的应用。

6.2.2 java.applet 包与 Applet 类

每个 Java 小程序都是 Applet 类的一个子类,换句话说,Applet 类是所有小程序的父类。Applet 类在 java.applet 包中定义,java.applet 包中除了定义 Applet 类外,还定义了其他内容。

在 Visual J++ 中查看 java.applet 包与 Applet 类所包含的内容,可以打开如图 6.10 所示的当前小程序的 Class Outline 窗口(打开 Class Outline 窗口的方法在 6.1 节中已经做过介绍)。

下面介绍 Applet 类中与 Applet 的生命周期相关的 4 个方法: `init()`、`start()`、`stop()` 和 `destroy()`。

Applet 的生命周期与 Web 页面有关。当打开包含 Applet 的 Web 页面时, Applet 的 `init()` 和 `start()` 方法被自动执行;当离开页面时, Applet 的 `stop()` 方法被自动执行;当重新加载页面、刷新页面或关闭浏览器时, Applet 的 `destroy()` 方法被自动执行。

一个 Applet 的生命周期如图 6.12 所示。



图 6.12 Applet 的生命周期

- ◆ `init()` 方法是对 Applet 进行初始化。当包含 Applet 的 Web 页面在浏览器中被首次打开时, Applet 被浏览器载入, 这时 Applet 的 `init()` 方法被调用。通过 `init()` 方法可以执行必要的启动代码, 例如变量初始化, 调入 Applet 所需的图像、声音资源等。
- ◆ `start()` 方法的作用是启动 Applet。包含 Applet 的 Web 页面在浏览器中每次被打开时, 都要调用 `start()` 方法使 Applet 启动。如果在 Applet 中创建新的线程, 一般是放在 `start()` 方法中实现。
- ◆ `stop()` 方法使 Applet 暂停或关闭。在浏览器中每次离开包含 Applet 的 Web 页面时, 都要调用 `stop()` 方法。
- ◆ `destroy()` 方法使 Applet 终止。当重新加载页面、刷新页面或关闭浏览器时, 都要调用 `destroy()` 方法, 释放分配给 Applet 的所有资源。

上面讲到的 4 种方法被调用的时机实际上与浏览器的类型有密切关系, 表 6.1 对 Navigator 和 Internet Explorer 两种浏览器在上述 4 种方法的调用方面进行对比。

表 6.1 Navigator 和 Internet Explorer 中调用 Applet 方法的比较

事件	Navigator	Internet Explorer
页首次打开	<code>init()</code> + <code>start()</code>	<code>init()</code> + <code>start()</code>
页前翻、页后翻	<code>Stop()</code> + <code>start()</code>	<code>stop()</code> + <code>start()</code>
页重载、页刷新	<code>stop()</code> + <code>start()</code>	<code>stop()</code> + <code>destroy()</code> + <code>init()</code> + <code>start()</code>

Java 小程序能够工作, 是因为载入 Java 小程序的浏览器知道调用它的哪个方法。Java 小程序都是 Applet 类的一个子类, Java 小程序从 Applet 类中继承了 Applet 类中的所有的方法, 而支持 Java 的浏览器知道 Applet 类中包含的方法, 因此也就知道 Java 小程序所继承下来的那些方法, 从而能够调用相应的方法使 Java

小程序工作起来。如果我们在 Java 小程序中创建了一个 Applet 类中没有的方法,那么浏览器将不知道这个新方法。虽然浏览器不能直接调用这些方法,但是我们可以使用重载(Override)的办法来使浏览器能够调用这些新的方法。

为了确定一个 Java 小程序中可用的所有方法,可以在 Object Browser 窗口中按照类的层次关系查看它们。图 6.13 显示了 Applet 类的层次关系,图中左边窗口显示的是 Applet 类的各级父类(Superclasses),图中右边窗口显示的是 Applet 类的直接父类——Panel 类的成员变量和方法。

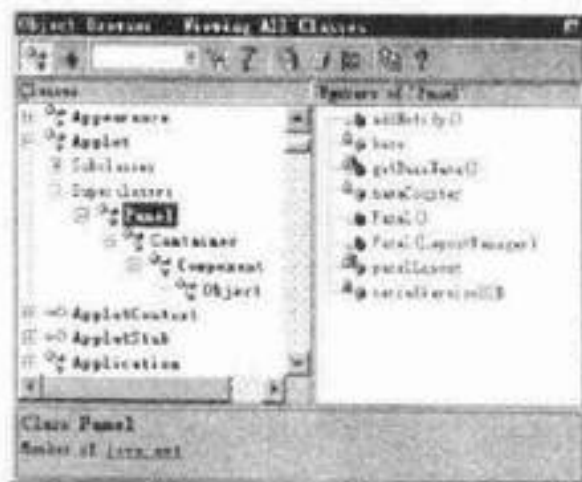


图 6.13 Applet 类的层次关系

6.3 java.awt 包

6.3.1 理解程序

在前面的程序中,我们除了引入 java.applet 包,还通过“import java.awt.*;”语句引入 java.awt 包。AWT 是 Abstract Window Toolkit 的缩写,意思是“抽象窗口工具箱”。引入 java.awt 包的目的是为了在程序中能够调用 java.awt 包中的 paint(Graphics)方法在屏幕上显示一段文字。

java.awt 包的作用是在连接到虚拟机的任何显示屏幕上画图。我们能够用 AWT 在 Web 网页上画一些简单的图形,例如文本、直线、圆、圆弧、长方形、多边形和其他图形,还可以绘制按钮、菜单、复选框等。在前面的例子中我们使用 AWT 在屏幕上显示的文字被认为是一个图形,而不是一个文本。从 6.1.4 小节我们知道,程序中文字的显示是通过重载 java.awt 包中 Container 类的 paint(Graphics)方法实现的,参看例子中所用的代码:



```
public void paint(Graphics pl)
{
    // TODO: Add your own implementation.
    super.paint(pl);
    pl.setColor(color.red);
    pl.drawString("欢迎来到 Java 的世界!", 50, 50);
}
```

上面代码中“super.paint(pl);”表示这里的 paint(Graphics)方法首先是继承父类 Container 的 paint(Graphics)方法。实际上 Container 类的 paint 方法是一个空方法,并不实现什么功能,只是提供一个方法声明来供子类继承。后面的语句“pl.drawString(“欢迎来到 Java 的世界!”, 50, 50);”则是为我们自己的 paint(Graphics)方法增加新的功能。

方法 drawString 是 Graphics 类中的方法,它是通过 Graphics 类的对象 pl 来调用的,格式是“pl.drawString()”。和 Container 类一样,Graphics 类也是定义在 java.awt 包中的。

6.3.2 java.awt 包中的类

java.awt 包是 Java 可用的包中最为庞大的一个包,它包括 60 多个有用的类。这些类允许您把用户界面建立在 Java 小程序中,例如 java.awt 包中有些类包含用于建立简单界面所需要的控制,如按钮、编辑框、复选框等等。而且 java.awt 包中还有几个不同的包容器用来安排屏幕的控制,如组件布局、网格布局等,并带有很多特性。下面我们只对 java.awt 包中比较常用的类做一个简要的介绍。

- ◆ **Container 类。**一般容器,可以在上面进行布局管理,其他所有的容器类都由它派生。
- ◆ **Panel 类。**Container 类的具体形态,没有图形表示,通常用 Panel 的子类来定义和控制组件的逻辑组合。
- ◆ **Button 类。**提供按钮,可以在按钮上加上文字标注。
- ◆ **Canvas 类。**提供画布,即一个可绘画区。它是不可见的,但可测出鼠标单击和移动事件。
- ◆ **Checkbox 类。**提供复选框。
- ◆ **Choice 类。**提供多项选择组件。
- ◆ **Label 类。**提供文字标注。
- ◆ **Scrollbar 类。**提供垂直或水平滚动条。
- ◆ **TextField 类。**提供单行窗口,用于输入文本。

- ◆ **BorderLayout** 类。边框布局,根据组件在窗口的分布规则确定组件的大小和位置,使组件沿着窗口的上、下、左、右边排放。
- ◆ **GridLayout** 类。网格布局,不用二维的坐标来为组件定位,而是通过一维的索引来定位。
- ◆ **Menu** 类。提供菜单,菜单的内容可以是菜单项或子菜单。
- ◆ **PopupMenu** 类。提供弹出菜单,单击鼠标右键显示的快捷菜单一般都是 **PopupMenu**。
- ◆ **Image** 类。提供一种图形数据结构,通常用于显示图片。
- ◆ **Font** 类。提供字体。
- ◆ **Color** 类。提供独立于平台的颜色数据结构。
- ◆ **Cursor** 类。指定鼠标光标的样子。

下面以建立按钮为例,简单介绍上面诸多类的用法。我们把按钮放到一个称为 **BorderLayout** 的容器中,在屏幕上按钮将显示在窗口的上边,并且按钮上显示文字“This is a button”。值得注意的是工程中 Java 源文件的名字要与这里的类名相对应,即 Java 源文件名应为 **ButtonTest.java**,否则下面的程序就不会编译成功。



```
import java.awt.*;
import java.applet.*;

public class ButtonTest extends Applet
{
    Button btn;
    public ButtonTest()
    {
        setLayout(new BorderLayout());
        btn = new Button("This is a button");
        add("North", btn);
    }
}
```

从上面的代码可以看到, **ButtonTest()** 方法名与小程序的类名相同。其实, **ButtonTest()** 是 **ButtonTest** 类的构造函数。构造函数用于类的初始化,这里初始化是完成设置边框布局、新建按钮、把按钮添加到布局中等工作。边框布局 **BorderLayout** 有 5 个附属点: **North**、**South**、**East**、**West** 和 **Center**。一旦附带了它们,就会依赖 **Applet** 在屏幕上的大小来处理对象的精确位置。“**add("North", btn);**”语句表示把按钮放在布局的 **North**(顶部)。



注意 **Applet** 类的父类——**Panel** 类定义在 **java.awt** 包中,从 **Applet** 类的层次关系(图 6.13)可以看到这一点。**java.awt** 包中绝大多数类的父类是 **Component** 类, **Applet** 类实际也是从 **Com-**

ponent 类到 Container 类再到 Panel 类一路继承下来的(如图 6.14 所示)。因此尽管 Applet 类在 java.applet 包中单独定义,但它的父类却是在 java.awt 包中定义的。

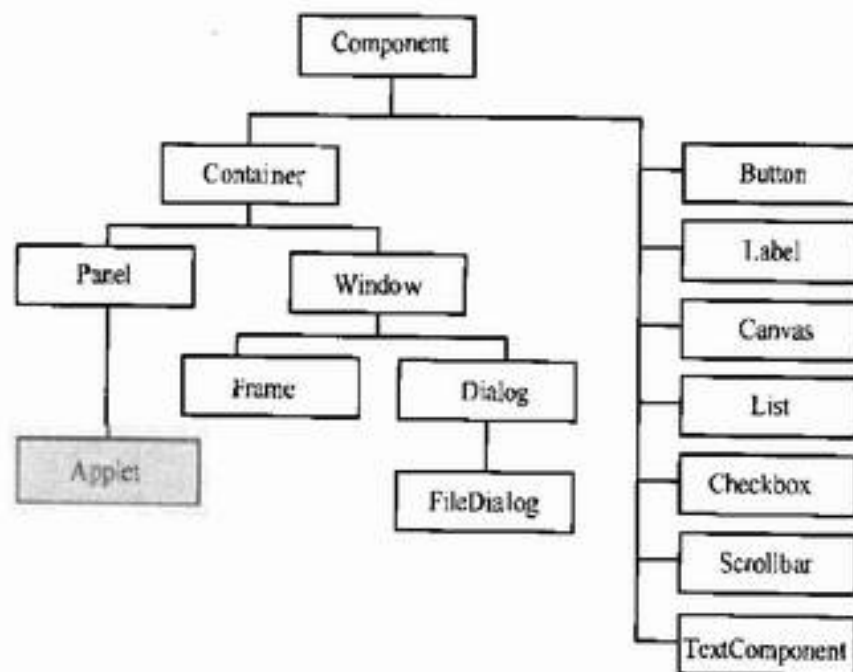


图 6.14 java.awt 包中 Component 类的继承关系

6.4 事件处理

Java 的跨平台性不仅适用于非交互程序,而且也适用于交互程序。对于图形用户界面来说,没用事件处理是不可想象的,Java 有它自己在 AWT 包内部的事件处理方式。在本节中,我们将介绍如何设计人机交互界面、如何接收用户的输入以及如何处理这些输入。

6.4.1 处理事件

一般情况下,事件是由用户的行为产生的,事件处理能够响应用户的命令,例如当用户按一个键或鼠标时,会产生事件并传给 Java 程序,程序必须处理并响应事件。

在 Java 环境中,所有的时间都作为 Event 对象传递,它们是 Event 类实例,Event 类封装了处理事件的所有信息。Event 类声明如下:

```
public class Event extends Object implements Serializable
```

Event 类是一个与平台无关的类,它为通过本地图形界面产生的事件提供了一个抽象层。Event 类是 AWT 的一部分,由 Object 类派生而来。

AWT 包有两个不同的事件模型。第一个是 Java 1.0 版的,第二个是 Java 1.1 版的。1.0 版模型比 1.1 版模型更小更简单,但它的功能也更少一些,而且不善于处理大的应用程序。Event 类向后兼容,因此 Java 程序员编程时要考虑这一点。

Event 类一般情况下是由 Java 环境自动创建和传递的,我们通过下列方法之一可以构造 Event 类:

- ◆ `public Event(Object target, long when, int id, int x, int y, int key, int modifiers, Object arg)`
- ◆ `public Event(Object target, long when, int id, int x, int y, int key, int modifiers)`
- ◆ `public Event(Object target, int id, Object arg)`

Event 类的构造方法需要目标组件和事件类型(id)作为参数,根据事件属性的不同,可能需要 x 坐标、y 坐标、键盘的按键、修饰键的状态以及事件的引用。

用户产生的事件需要 Java 程序处理,一个事件的处理包含两个方面:

- ◆ 识别发生的事件类型
- ◆ 调用正确的 Java 代码处理事件

Java 环境为常用的事件提供了默认的处理方法 `handleEvent()`,即事件处理器(Event Handler)。该方法由 Component 类提供,因为所有的 UI 组件都是由 Component 类派生而来,它们都继承了 Component 类的事件处理方法。事件处理器负责接收发生的事件,并对事件作出处理。事件处理器的调用方式是:

```
public boolean handleEvent(Event evt)
{
    |
    |
}
```

在事件处理器中根据 Event 类参数 evt 的 id 值来判断发生事件的种类,针对用户界面(UI)构件的事件、键盘的按键、鼠标的操作、窗口的创建、移动、撤销等等。在某种情况下如果需要重载 `handleEvent` 方法,应相当慎重。因为对于原来的这些事件处理方法,除非在重载后的方法中加以指出,否则是不会被调用的。比较可靠的重载方法是:

```
public boolean handleEvent (Event evt) {
    if (evt.id == Event.KEY_PRESS) {
        .....
        return true; //处理键盘操作
    }
}
```



```

        else {
            return super.handleEvent(evt);
        }
    }
}

```

这里是将余下的事件交给 `handleEvent` 的父类处理,从而保证对事件的正常处理。

6.4.2 处理常用事件

Java 中的事件(Event)处理是 AWT 的一部分。事件发生时,通过这些事件,AWT 构件与用户或 AWT 构件之间就可以进行某种通信。AWT 对事件的处理分为两种情况,一种是由 AWT 或浏览器负责处理,例如 `paint()` 方法;还有一种是不作处理,例如鼠标的移动。在这里,我们要讨论的是对于那些 AWT 不作处理的方法,如何编写并重载事件处理方法,例如接收鼠标、键盘的输入。关于 AWT 如何处理这些事件,在本节的最后一部分将专门讨论。为了接收鼠标、键盘的输入,可以把它们相应的事件分成 3 类:鼠标的按键、鼠标的移动或拖动、键盘的输入。

鼠标的按键

对于鼠标的按键,AWT 会产生两种事件。当鼠标左键按下时产生 `mouseDown` 事件,当鼠标左键弹起时产生 `mouseUp` 事件。对于这两种事件,系统会分别调用相应的处理方法 `mouseDown()` 和 `mouseUp()`。所以为了在程序中接收这两种事件,需要在程序中重载这两个事件的处理方法。例如对 `mouseDown()` 方法,其调用方式为:



```

public boolean mouseDown(Event evt, int x, int y)
{
    ...
}

```

其中 `X`、`Y` 是事件发生时鼠标的位置, `evt` 参数是由系统产生的 `Event` 类的一个实例,它包含了关于这个事件的一些信息。

`mouseUp()` 的使用与 `mouseDown()` 相同。下一小节,我们使用此方法完成输出的中英文切换。

鼠标的移动

鼠标的移动有两种情况:鼠标的移动和鼠标的拖动。同样也对应了两个方

法: `mouseDrag` 和 `mouseMove`。其调用方式和使用方法与 `mouseDown` 相同:



```
public boolean mouseDrag( Event evt, int x, int y)
|
|
public boolean mouseMove(Event evt, int x, int y)
|
|
```

键盘的输入

当用户按下或松开键盘的按键时,会产生键盘事件。相应的处理方法是 `keyDown` 和 `keyUp`。它们的调用方式与上面的略有不同。

```
public boolean keyDown (Event evt, int key)
|
|
```

这里的 `key` 参数是一个整数变量,是所按下的键的 ASCII 值。可以用类型转换 `(char)key` 将其转成字符类型。`keyUp` 的调用方式相同。这两种方法的区别是使用场合不同。如果将按住键不放解释为一连串相同的输入时,用 `keyDown` 来处理;而将按住键不放解释为一个字符输入时,用 `keyUp` 来处理。另一个问题是对于某些特殊键如何处理,如 `Home`、`End`、`PageUp`、`PageDown` 和方向键等,Java 中提供了较为简单的方法,它将这些特殊 ASCII 值定义成 `Event` 类的变量。例如判断是否为 `Home` 键,可以写成:

```
if (key == Event.HOME)
|
|
```

这样在编写程序时,就方便多了。Java 的 `Event` 类还另外提供 3 个方法 `shiftDown()`、`metaDown()` 和 `controlDown()` 来判断 `Shift`、`Alt`、`Ctrl` 键是否按下。使用方法如下:



```
public boolean keyDown (Event evt, int key) {
    if (evt.shiftDown())
        .....
    else
        .....
}
```

6.4.3 在小程序中添加事件处理代码

接下来我们看看具体的例子,将 ex06a 目录复制到 ex06b 目录中,在 ex06b 目录下打开工程,另存工程和解决方案为 ex06b。对于 ex06b 例子,我们决定要截取鼠标事件,当单击鼠标时输出改为英文“Welcome to Java World!”,再单击鼠标输出改为“欢迎来到 Java 的世界!”,即鼠标事件完成中英文切换功能。为了找到要重载的正确事件处理方法,查看以 mouse 开头的所有事件会有所帮助。这里我们要处理的事件是释放鼠标按钮事件,需要重载的方法是 mouseUp。下面是 Visual J++ 在重载 mouseUp 方法后自动生成的代码:



```
public boolean mouseUp(Event p1, int p2, int p3)
{
    // TODO: Add your own implementation.
    return super.mouseUp(p1, p2, p3);
}
```

在小程序中添加这个新方法并运行它。在 Web 浏览器的小程序显示区域中单击鼠标时,并没有什么变化,这是因为我们没有在 mouseUp 方法中进行相应的处理。为了实现中英文切换功能,需要定义一个成员变量 language,当 language 为“english”时,单击鼠标输出英文;当 language 为“chinese”时,单击鼠标输出中文。下面我们看看怎样给应用程序添加成员变量。

1. 在 Class Outline 面板中选择 Class1,单击鼠标右键,显示快捷菜单如图 6.15 所示。
2. 选择 Add Member Variable 命令,显示添加成员变量出口,如图 6.16 所示。

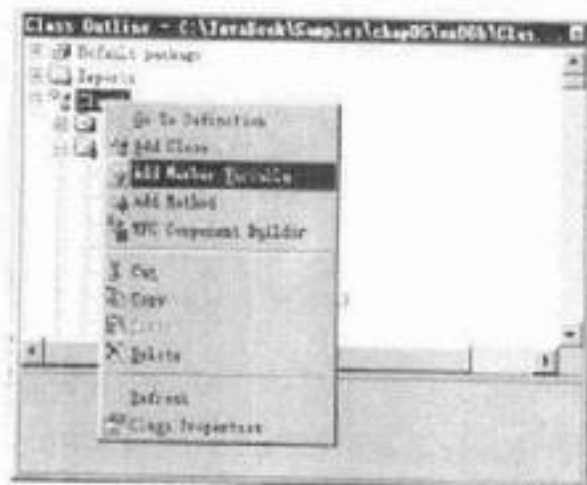


图 6.15 带快捷菜单的 Class Outline 面板



图 6.16 添加成员变量

3. 在 Name 编辑框中键入“language”，在 Data Type 下拉框中选择 java.lang.String，在 Access 下拉框中选择 Private，在 Modifiers 组合框中选择 Static，在 Initial Value 编辑框中键入“chinese”，然后单击 Add 按钮，Visual J++ 自动生成如下代码：

```
private static String language = "chinese";
```

当然，我们也可以手工键入上面的代码。有了 language 这个成员变量，我们修改 paint 方法，根据 language 成员变量的值判断是输出中文还是英文，修改后的代码如下：



```
public void paint(Graphics pl)
{
    // TODO: Add your own implementation.
    super.paint(pl);
    if(language.equals("english")) {
        pl.drawString("Welcome to Java World!", 50, 50);
    }
    else {
        pl.drawString("欢迎来到 Java 的世界!", 50, 50);
    }
}
```

equals 方法判断 language 是否等于“English”，结果是输出英文，否则输出中文。这样我们在 mouseUp 方法中调用 repaint 方法，对 repaint 方法的调用使我们可以看到显示区域的改变。因为直到显示屏幕被刷新，输出的改变才能看得出

来。调用 `repaint` 会在内部调用 `paint`, 因此小程序的外观会改变。

`mouseUp` 方法的完整代码如下:



```
public boolean mouseUp(Event p1, int p2, int p3)
{
    // TODO: Add your own implementation.
    repaint();
    if(language.equals("english"))
        language = "chinese";
    else
        language = "english";

    return true;
    //return super.mouseUp(p1, p2, p3);
}
```

注意, 我们不能在 `mouseUp` 方法中直接调用 `paint`, 因为它需要 `Graphics` 型参数值, 而在我们的 `mouseUp` 事件中没有 `Graphics` 型参数; 然后我们改变 `language` 的值, 这样就完成了中英文切换的功能; 最后, 我们返回布尔值 `true`, 删除 `return super.mouseUp(p1, p2, p3)` 语句。有一点需要说明的是, 这个方法必须返回一个布尔值。返回什么样的布尔值, 取决于方法对事件的处理。如果返回 `true`, 则表明方法已完成对事件的处理; 如果返回 `false`, 则表明需要其他 AWT 构件来处理。在大多数情况下, 都是返回 `true`。

运行 Java 小程序, 结果如前面图 6.9 (与 `ex06a` 的运行结果一样), 单击鼠标输出变为英文, 如图 6.17 所示。



图 6.17 `ex06b` 的英文显示

6.5 把小程序嵌入到 Web 页

通过前面的学习我们已经熟悉了 `Applet` 类, 所有的 Java 小程序都是由它派

生而来。除此之外,我们还了解了 Java 小程序和它所在环境的交互方式。编写 Java 小程序的主要目的是在 Web 网页中使用,那么如何在 Web 网页中使用小程序呢?这一节我们主要介绍在 Web 网页中使用小程序。

6.5.1 向工程中加入 Web 网页

运行小程序需要 Java 小程序浏览器。最常用的 Java 小程序浏览器是 Web 浏览器,例如 Internet Explorer 和 Navigator。由于 Web 浏览器起着一个 Java 小程序浏览器的作用,所以它必须支持 Java。从原理上讲,Java 小程序可以运行在任何计算机上,只要该计算机有 Java 的虚拟机和某种 Java 小程序浏览器。我们前几节的例子都是使用 WJView 运行小程序,这种方式不能使用 Java 小程序的参数,我们将在下一节讲如何使用 Java 小程序参数。使用简单一些的 Java 小程序浏览器,也可以不用打开 Web 浏览器,就可以快速浏览 Java 小程序。我们很快就可以看到,Visual J++ 提供了这种能够使用参数的简单的 Web 浏览器——Quick View。下面我们看看生成 Web 网页的详细步骤。

► 生成 Web 网页

1. 在 Project Explorer 窗口中右键单击工程图标,并在显示的快捷菜单中指向 Add,然后选择 Add Web Page 命令,显示 Add Item 对话框,如图 6.18 所示。

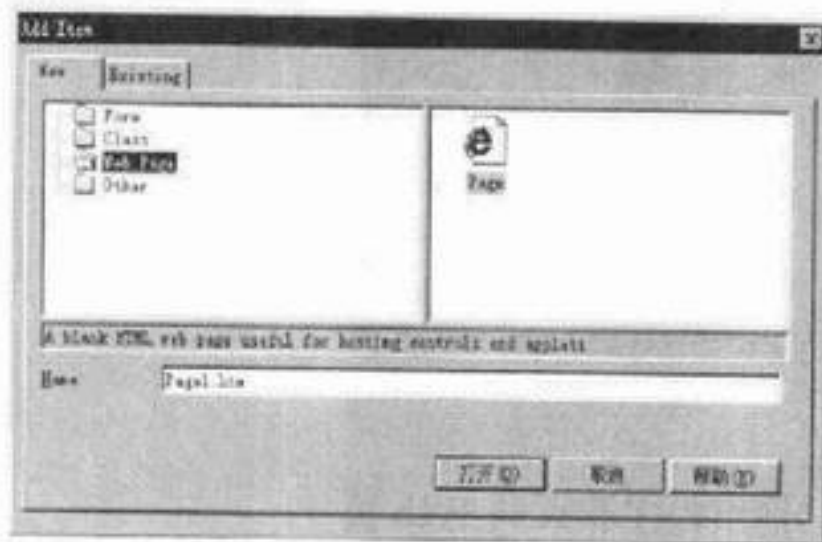


图 6.18 Add Item 对话框

2. 在 Add Item 对话框的左边栏中选定 Web Page。在右边的一栏中选择 Page，使用默认的 Page1.htm 作为名称。
3. 单击【打开】按钮，Visual J++ 会自动打开 page1.htm，如图 6.19 所示。



图 6.19 HTML 文档的代码窗口

此时我们完成了添加 Web 网页的功能，图 6.19 HTML 文档的代码窗口显示的是 Design 选项卡。选择菜单 View 下的 Toolbox 子菜单，会出现 Toolbox 面板，如图 6.20，这时我们可以直接向 HTML 文档的代码窗口中添加 Toolbox 面板的各种 HTML 元素。单击 Source 选项卡，会看到系统已经自动生成了一个 HTML 框架：



```
< HTML >
< HEAD >
< META NAME = "GENERATOR" Content = "Microsoft Visual Studio 6.0" >
< TITLE> </TITLE>
</HEAD>
< BODY >
< P> &nbsp;</P>
</BODY>
</HTML>
```



图 6.20 Toolbox 面板

在 <TITLE> 和 </TITLE> 间加入“ex06c — 在 Web 网页中使用小程序”。

用 <APPLET> 标记,在代码的 <P> 和 </P> 行后面分别添加 <APPLET> 和 </APPLET> 标记,HTML 代码如下:



```
< HTML >
< HEAD >
< META NAME = "GENERATOR" Content = "Microsoft Visual Studio 6.0" >
< TITLE > ex06c — 在 Web 网页中使用小程序 </TITLE>
</HEAD>
< BODY >
< P> &nbsp;&nbsp;&nbsp;</P>
< APPLET
    NAME = Class1
    CODE = Class1.class
    WIDTH = 320
    HEIGHT = 180 >
</APPLET>
</BODY>
</HTML>
```



注意 在 <APPLET> 和 </APPLET> 标记之间的文本,只有在浏览器不支持 Java 或 Java 支持被禁用时才显示。如果浏览器不支持 Java,则不显示小程序。这时,小程序只显示背景,它是灰色的,为了迎合不支持 Java 的浏览器,可以在 <APPLET> 和 </APPLET> 标记之间加入说明“你的浏览器不支持 Java”以提醒 Web 用户。单击 Quick View 选项卡,看看我们的 Java 小程序是什么样子,如图 6.21 所示。



图 6.21 Quick View 窗口

再次选择 Source 选项卡, 结果如图 6.22 所示。



图 6.22 HTML 源码图形显示方式

如果查看源码文本方式, 选择 View 菜单下的 View controls As Text 子菜单, HTML 源码窗口显示如图 6.23 所示。



图 6.23 HTML 源码文本显示方式

至此我们完成了向工程中添加 Web 网页,在 Web 网页中使用 Applet。您也许不能完全理解上面的步骤,那么让我们了解一下 HTML 标记的知识。

6.5.2 基本 HTML 标记

HTML 是超文本标记语言的简写,它不是一种编程语言,而是一种页面布局语言。传统上,HTML 用于告诉 Web 浏览器在页面中包含的文本和图形。HTML 由各种标记组成,标记是关于 Web 网页显示的指令。HTML 文件被 Web 浏览器读取,产生我们看到的网页的文件。从本质上来说,Internet 只是一个由 HTML 文件及一系列传输协议所组成的集合。这些 HTML 文件存储在分布于世界各地的计算机硬盘上,而传输协议能把这些文件从一台计算机传输到另一台计算机。Web 浏览器,例如 Navigator 或 Internet Explorer,能够解释 HTML 文件从而显示网页,这是 Web 浏览器的主要作用。当使用浏览器在 Internet 上浏览网页时,浏览器软件就自动完成 HTML 文件到网页的转换。HTML 标记能够很容易被识别出来,因为它们以尖括号(< >)开始和结束。有些 HTML 标记在尖括号中也包含了 HTML 属性,这些属性允许我们给 HTML 标记指定特定的值。下面我们以前面的 Page1.htm 的 HTML 代码来理解 HTML 标记,分别讲解各行 HTML 代码:

◆ <HTML>

这个标记把它后面的代码当作应该用来解释显示的 HTML 信息。在文件的末尾是 </HTML>,它表示 HTML 的结束。许多标记用同样的格式来开始和结束。这就是说,一个关闭标志指示特定的 HTML 标记指令的结束。<HTML> 标识符说明了这个文件是一个 HTML 文件。这个标识符并不是必须的,但对于旧版的浏览器,应当在文件头部包含 <HTML> 标识符,并在文件尾包含 </HTML> 标识符来确保兼容性。

◆ `</HTML>`

这个标记结束 HTML 文档。表 6.2 列出了基本的 HTML 标记。

表 6.2 创建 Web 页面所需的基本 HTML 标记

标记	含义
<code><html></code>	HTML 页面开始
<code></html></code>	HTML 页面结束
<code><title></code>	标题开始
<code></title></code>	标题结束
<code><body></code>	HTML 文件的主体部分开始
<code></body></code>	HTML 文件的主体部分结束

解释完 Page1.htm, 我们详细讲述 `<APPLET>` 标记, 它是 Java 小程序中最重要的 HTML 标记。

6.5.3 `<APPLET>` 标记

我们看到了 Page1.htm 中正在起作用的 `<APPLET>` 标记。这是小程序中最重要的 HTML 标记, `<APPLET>` 标记用于定义一个嵌入的 Java 小程序, 它允许小程序运行在 Web 网页内部。需要指出的是作为一个定界符, 一个 `<APPLET>` 标记必须和 `</APPLET>` 标记成对使用。事实上, 只标记 `<APPLET>` 而没有标记 `</APPLET>`, Java 小程序不会执行, 读者不妨试一试。

可以用 `<APPLET>` 标记的属性来制定 Java 小程序的特征, 如 Java 小程序的可执行文件的位置以及小程序显示的大小和位置。`<APPLET>` 标记的主要属性如表 6.3 所示。

表 6.3 `<APPLET>` 标记的主要属性

属性	说明
CODEBASE	代码库
CODE	代码
NAME	名字
ID	ID 号
WIDTH	宽度
HEIGHT	高度
ALIGN	对齐
HSPACE	水平空间
VSPACE	垂直空间

◆ **CODEBASE = URL**

这个属性指定小运行程序的基 URL。如果这个特性没有被指定,那么这个小运行程序的基 URL 与当前文档相同。如:



```
< applet CODE = MyApplet.class CODEBASE = Appletdir >
```

CODEBASE 属性指明了名为 MyApplet.class 的 Java 类文件在 Appletdir 目录下。

◆ **CODE = cdata**

这个属性指定包含小运行程序编译子类的资源的名称。这个值必须是一个基于小运行程序基 URL 的相对 URL。“ .class”后缀可以省略,Java 环境自动寻找 class 文件。

◆ **NAME = cdata**

这个属性指定了 Java 小程序实例的名称,这就使 Java 小程序在同一页找到(或与之通信)其他 Java 小程序变为可能。主要用于 Javascript, Javascript 用 name 属性来标识 Java 小程序。

◆ **ID**

这个属性指定 Java 小程序的名字,它能被 VBScript 引用,通过 VBScript 使用,可以控制小程序。

◆ **WIDTH = length**

这个属性指定了 Java 小程序显示区域的初始宽度,单位为像素。

◆ **HEIGHT = length**

这个属性指定了 Java 小程序显示区域的初始高度,单位为像素。

◆ **ALIGN = BOTTOM|MIDDLE|TOP|LEFT|RIGHT**

这个属性指定了物件与周围上下文的位置关系。表 6.4 包含了此属性的可能值和效果。

表 6.4 ALIGN 可能值和效果

对齐值	效果
LEFT	将 Java 小程序放在左边空白区域
RIGHT	将 Java 小程序放在右边空白区域
TOP	将 Java 小程序与行中最靠上的部分对齐
TEXTTOP	将 Java 小程序的顶部与行中最高的字符对齐
MIDDLE	将 Java 小程序与相邻文本的基线的中部对齐
ABSMIDDLE	将 Java 小程序的顶部与行中最高的字符的中部对齐
BASELINE	将 Java 小程序与相邻文本的基线对齐
BOTTOM	将 Java 小程序与相邻文本的基线的低部对齐
ABSBOTTOM	将 Java 小程序与行中最低的部分对齐

◆ HSPACE = length

这个属性指定了 Java 小程序与左边或右边显示的文本之间的像素数目。

◆ VSPACE = length

这个属性指定了 Java 小程序与上、下的文本之间的像素数目。

◆ ALT = 替代文字

这个属性指定了不支持 Java 或 Web 浏览器中 Java 被禁用时显示的字符串。

表 6.5 列出了 <APPLET> 标记属性的摘要。注意 HTML 标记不区分大小写。这里按照习惯把它们写成大写。

表 6.5 HTML 中 <APPLET> 标记属性概览

属性	用途	值	是否必需
CODEBASE	文件的可选位置	URL	否
CODE	Java 类文件的文件名	Java 类文件名	是
NAME	运行小程序的名称	字符串	否
ID	运行小程序的标识	字符串	否
HEIGHT	小程序显示区域的垂直高度	数字	是
WIDTH	小程序显示区域的水平宽度	数字	是
HSPACE	小程序周围的水平空格数	数字	否
VSPACE	小程序周围的垂直空格数	数字	否
ALIGN	网页上小程序的布置	Left、Right、Top、Middle 或 Bottom	否
ALT	Web 浏览器中 Java 被禁用时显示的字符串	字符串	否

用一个或多个 <PARAM> 标记可以从 HTML 文件向 Java 小程序传递参数, <PARAM> 标记必须出现在 <APPLET> 标记的内部, <PARAM> 标记的属性是 name 和 value。

6.6 使用 Applet 参数

Applet 非常灵活,在 Applet 启动时通过向 Applet 传递参数可以定制它。传递参数的关键是在 HTML 中使用 <PARAM> 标记,Applet 处理这些参数。通常一个好的小程序的关键在于灵活性,如果通过引进参数,允许别人把我们的 Applet 嵌入到他们的文档中,并能允许他们自己定义小程序代码的一些行为或特征,那么我们可以使 Applet 在 WWW 被更多的人使用。

传递 Applet 参数很容易实现。在本节中我们扩展本章的例子,增加新的功

能。通过向 Applet 传递“language”参数,控制 Applet 初始状态显示的语言是英文还是中文。

6.6.1 <PARAM> 标记

在 <APPLET> 和 </APPLET> 加入 <PARAM> 标记传递 Applet 参数,<PARAM> 标记格式为:

```
<PARAM NAME= name VALUE= value>
```

其中 NAME 为参数名,VALUE 为参数值,参数值为字符串。修改 ex06c 例子中的 Pugel.htm 文件,把 Applet 标记改为:



```
<applet  
    code= Applet1.class  
    name= Applet1  
    width= 320  
    height= 180 >  
    <param name= mylanguage value= "chinese" >  
</applet>
```

仅仅这样做是不够的,读者试着运行程序,并没有什么改变。这是因为虽然我们传递了参数,但并没有处理这些参数。要使小程序能使用相关值,小程序代码必须知道参数名(即 NAME 属性的值)。虽然参数名是在小程序代码中预先确定的,但 VALUE 可以是任意字符串。作为程序员,应该用小程序给别人提供一个可用参数列表。下面我们看看在 Applet 中怎样使用 mylanguage 参数。

6.6.2 在小程序中使用参数

我们的下一步是在小程序中编写代码,从 HTML 文件中读取参数值。将 ex06c 目录复制到 ex06d 目录中,在 ex06d 目录下打开工程,另存工程和解决方案为 ex06d。在 Applet 中读取参数的方法为 getParameter 方法。在 6.2 节中我们了解了 Java 小程序的生命周期,而 getParameter 方法通常是在小程序的 init 方法中被调用。GetParameter 的语法格式为:

```
public String getParameter( String name )
```

参数说明

name

参数名

返回值

参数值

让我们看看具体的例子,按着前面的步骤在类 Class1 加入 init 方法,自动生成的代码如下:

```
public void init()
{
    // TODO: Add your own implementation.
    super.init();
}
```

我们用 String 参数来调用 getParameter 方法,这个参数的值同 HTML 文件中相应的 <PARAM> 标记的 NAME 属性值相匹配。该方法返回与 NAME 相关的 VALUE 属性的值。本例中,因为我们给 NAME 的值为 mylanguage,它返回字符串 "english",并把该值赋给成员变量 language:

```
language = getParameter("mylanguage");
```

最好检验是否返回了值,并为没有返回值的情况提供值:

```
if(language == null)
    language = "chinese";
```

如果成员变量 language 等于 null,要么该参数没有名字,即 <PARAM> 标记没有被包括在 HTML 文档中;要么是小程序可能是从一个没有解释 HTML 代码的小程序浏览器中执行的。任何情况下,如果 language 等于 null,则我们希望提供默认值。这样,完整的 init 方法代码为:

```
public void init()
{
    // TODO: Add your own implementation.
    super.init(); language = getParameter("mylanguage");
    if(language == null)
        language = "chinese";
}
```

为了让 Java 小程序的语言参数更为灵活,在比较字符串前,使用 toLowerCase 方法,把成员变量 language 变为小写字符,这样在 HTML 页面中 mylanguage 参数值不区分大小写。修改后的 mouseUp 事件和 paint 方法如下:



```
public void paint(Graphics g1)
{
    // TODO: Add your own implementation.
    super.paint(g1);
    language.toLowerCase();

    if(language.equals("english")) {
```

```
        pl.drawString("Welcome to Java World!",50,100);
    }
    else {
        pl.drawString(" 欢迎来到 Java 的世界!",50,100);
    }
}

public boolean mouseUp(Event p1, int p2, int p3)
{
    // TODO: Add your own implementation.
    //return super.mouseUp(p1, p2, p3);
    language.toLowerCase();
    if(language.equals("english"))
        language = "chinese";
    else
        language = "english";
    repaint();
    return true;
}
```

按 F5 键运行程序,在 Applet View 中看不到参数的结果,需要在浏览器中查看结果。在 Project Explore 面板中双击 page1.htm 文件,选择 Quick View 页,结果如图 6.24 所示。



图 6.24 page1.htm 浏览结果

为了设置 Applet 的语言参数为英文,另存 page1.htm 为 Mypage.htm,修改

```
<param name = mylanguage value = "chinese">
```

为

```
<param name = mylanguage value = "english">
```

保存 Mypage.htm,用 Quick View 查看,结果如图 6.25 所示。



图 6.25 page1.htm 浏览结果

6.7 精彩实例

这一节我们演示几个精彩实例,只要把它们从光盘复制到您自己的硬盘上,并且去掉所有文件的只读属性,就可以用 Visual J++ 打开相应的工程文件,运行程序,观看结果,相信您一定会感兴趣的。

在 Visual J++ 中运行本节的程序实例之前,注意在 Project 菜单中选择 Properties 命令,打开工程属性窗口,按照图 6.26 所示设置工程属性,这样才会在 IE 浏览器中查看结果——TEST.HTML。

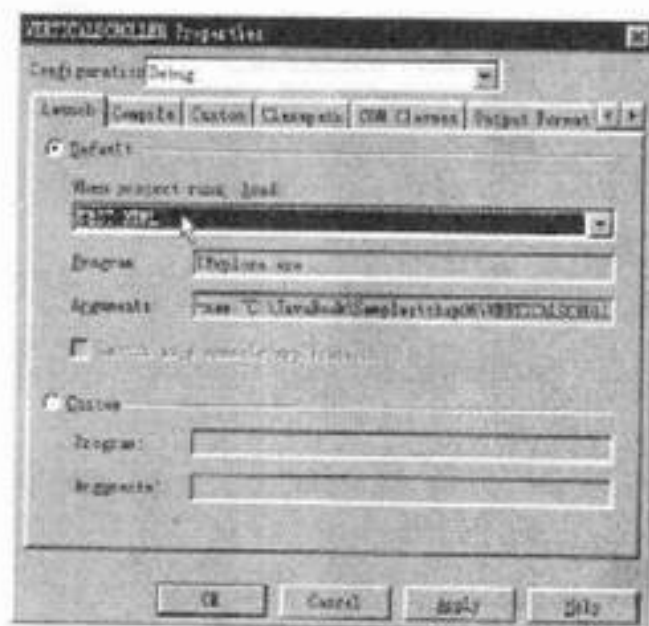


图 6.26 设置工程属性

6.7.1 滚动字幕

这是一个显示滚动字幕的 Applet 程序(如图 6.27),屏幕上显示的文字将一行一行地向上滚动,如果在 Applet 程序显示范围内单击鼠标,滚动将会停止,再次单击,滚动将继续。



图 6.27 滚动字幕

使用者可以借助对参数字串的输入,达到在网页上显示信息的目的,并可以继续加入更多的信息。以下是 TEST.htm 文件的 HTML 的范例:



```
<html>
<body>
<applet code="ShowLine.class" width=280 height=29>
  <param name="text0" value="白日依山尽 黄河入海流">
  <param name="text1" value="欲穷千里目 更上一层楼">
  <param name="text2" value="">
  <param name="text3" value="夜黑雁飞高 单于夜遁逃">
  <param name="text4" value="欲将轻骑逐 大雪满弓刀">
  <param name="text5" value="">

```



```

        </applet>
    </body>
</html>

```

源程序如下：



```

import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class ShowLine extends Applet implements Runnable, MouseListener{
    private Image buffer;
    private Graphics gContext;
    private String[] text;
    private String temp;
    private int numtext,currenttext;
    private boolean execute;
    private Thread thread;

    public void init(){
        int index,rgb;
        buffer = createImage(280,29);
        gContext = buffer.getGraphics();
        gContext.setColor(Color.black);
        gContext.fillRect(0,0,280,29);
        gContext.setFont(new Font("TimesRoman",Font.BOLD,12));
        index = 0;
        temp = getParameter("text" + String.valueOf(index));

        while(temp != null){
            ++ index;
            temp = getParameter("text" + String.valueOf(index));
        }

        numtext = index;
        text = new String[numtext];

        for(index = 0;index < numtext; ++ index){
            text[index] = new String(getParameter("text" + String.valueOf
(index)));
        }

        currenttext = 0;
        addMouseListener(this);
    }

    public void start(){
        if(thread == null){

```

```

        currenttext = 0;
        gContext.setColor(Color.black);
        gContext.fillRect(0,0,280,29);
        repaint();
        thread = new Thread(this);
        thread.start();
        execute = true;
    }

    public void stop(){
        if(thread != null){
            currenttext = 0;
            gContext.setColor(Color.black);
            gContext.fillRect(0,0,280,29);
            thread.stop();
            execute = false;
            thread = null;
        }
    }

    public void update(Graphics g){
        g.drawImage(buffer,0,0,this);
    }

    public void repaint(Graphics g){
        g.drawImage(buffer,0,0,this);
    }

    public void paint(Graphics g){
        g.drawImage(buffer,0,0,this);
    }

    public void run(){
        int index;
        int x;

        while(thread != null && execute == true){
            gContext.setColor(Color.white);

            for(index = 0; index < text[currenttext].length() + 1; ++ index){
                x = getSize().width - text[currenttext].length() * 12 ;
                System.out.println(x / 2);
                gContext.drawString(text[currenttext].substring(0, index), x/2, 26);
                repaint();
                try{
                    thread.sleep(50);
                } catch (InterruptedException e) {}
            }
        }
    }

```

```
        |
        |
        |currenttext += 1;
        |if(currenttext >= numtext)
        |    currenttext = 0;
        |try|
        |    thread.sleep(2000);
        |catch(InterruptedException e){|
        |
        |for(index = 0; index < 14; ++ index){
        |    gContext.copyArea(0,1,280,29,0, -1);
        |    repaint();
        |    try|
        |        thread.sleep(50);
        |    catch(InterruptedException e){|
        |
        |    }
        |}
        |
        |
        |public void mouseClicked(MouseEvent e)
        |
        |
        |
        |public void mousePressed(MouseEvent e)
        |
        |    if(execute == true){
        |        thread.suspend();
        |        execute = false;
        |    }else{
        |        thread.resume();
        |        execute = true;
        |    }
        |
        |
        |public void mouseReleased(MouseEvent e){
        |
        |
        |public void mouseEntered(MouseEvent e){
        |
        |
        |public void mouseExited(MouseEvent e){
        |
        |
        |
```

6.7.2 水波倒影

这是一个能够产生出图片水中倒影(如图 6.28),倒影并带有水波涟漪的 Applet。



图 6.28 水波倒影

使用者可以在 HTML 中设定自己喜欢的图形,从而可以产生一幅美丽的海岸风光。以下是 TEST.htm 文件的 HTML 的范例:



```
<html>
<body>
  <applet code=Ripple.class width=300 height=320>
    <param name= image value="Sunset.jpg">
  </applet>
</body>
</html>
```

HTML 需设定的参数有一个,即使用者欲显示出的图片,这里的参数值为“Sunset.jpg”,使用者在例子目录下可以加入自己的图片,更改 TEST.htm 文件中的参数后,就能看到自己的图片在水波倒影中的效果了。

源程序如下:



```
import java.applet.*;
import java.awt.*;

public class Ripple extends Applet implements Runnable
{
```

```
Thread thread = null;
private Graphics g, refraction;
private Image image, refimage;
private int currentImg;
private int imageW = 0, imageH = 0;
private int ovalW = 0, ovalH = 0;
private boolean finishLoad = false;
private final int frames = 12;
private String name = "";

public void init(){
    String param;
    param = getParameter("image");
    if (param != null)
        name = param;
}

public void paint(Graphics g){
    if (! finishLoad)
        return;

    if (refimage != null) {
        g.drawImage (refimage, (-currentImg * imageW), imageH, this);
        g.drawImage (refimage, ((frames-currentImg) * imageW), imageH, this);
    }
    g.drawImage (image, 0, -1, this);
}

public void start(){
    if (thread == null){
        thread = new Thread(this);
        thread.start();
    }
}

public void run(){
    currentImg = 0;
    g = getGraphics();
    MediaTracker imageTracker = new MediaTracker(this);
    String strImage;
    image = getImage(getDocumentBase(), name);
    imageTracker.addImage(image, 0);
    try{
        imageTracker.waitForAll();
        finishLoad = ! imageTracker.isErrorAny();
    }
    catch (InterruptedException e) {}
}
```

```

        imageW = image.getWidth(this);
        System.out.println(imageW);
        imageH = image.getHeight(this);
        createRipple();
        repaint();
        while (true) {
            try {
                if (! finishLoad)
                    return;

                if (refimage != null) {
                    g.drawImage (refimage, (-currentImg * imageW), imageH,
                                this);
                    g.drawImage (refimage, ((frames-currentImg) * imageW),
                                imageH, this);
                }
                g.drawImage (image, 0, -1, this);

                if (++currentImg == frames)
                    currentImg = 0;
                Thread.sleep(50);
            } catch (InterruptedException e) {
                stop();
            }
        }
    }

    public void createRipple () {
        Image back = createImage (imageW, imageH + 1);
        Graphics offg = back.getGraphics();
        int phase = 0;
        int x, y;
        double pl;
        offg.drawImage (image, 0, 1, this);

        for (int i = 0; i < (imageH >> 1); i++) {
            offg.copyArea (0, i, imageW, 1, 0, imageH - i);
            offg.copyArea (0, imageH - 1 - i, imageW, 1, 0, -imageH + 1 +
                           (i << 1));
            offg.copyArea (0, imageH, imageW, 1, 0, -1 - i);
        }

        refimage = createImage ((frames + 1) * imageW, imageH);
        refraction = refimage.getGraphics();
        refraction.drawImage (back, frames * imageW, 0, this);

        for (phase = 0; phase < frames; phase++) {
            pl = 2 * Math.PI * (double)phase / (double)frames;

```

```

x = (frames - phase) * imageW;
for (int i = 0; i < imageH; i++) {
    y = (int)((imageH/14) * ((double)i + 28.0)
        * Math.sin(((double)((imageH/14) * (imageH - i))/(double)(i + 1) + pi) / (double) imageH);
    if (i < -y)
        refraction.copyArea (frames * imageW, i, imageW, 1, -x, 0);
    else
        refraction.copyArea (frames * imageW, i + y, imageW, 1, -x,
            -y);
}
offg.drawImage (image, 0, 1, this);
image = back;
}
}

```

6.7.3 小丸子时钟

这是一个小丸子时钟的 Applet(如图 6.29),它可以读出当前时间,并以时钟指针的样子表现出来,指针随着时间不断移动,就像真的时钟一样。



图 6.29 小丸子时钟

这个例子不需要任何参数。以下是 TEST.htm 文件的 HTML 的范例:



```

<html>
<body>
<applet code="Clock.class" width="150" height="150">
</applet>
</body>
</html>

```


这个例子的源程序由两个 .java 文件组成: Clock.java 和 RotateG.java。
源程序如下:



```
//Clock.java
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.text.*;
import java.net.*;

public class Clock extends Applet implements Runnable
{
    private Date date;
    private int iHour,iMinute,iSecond;
    private Thread thread = null;
    private Image offI,backG;
    private Graphics offG;
    private int ixHour[] = {4,4,0,-4,-4}
        ,iyHour[] = {0,-30,-35,-30,0}
        ,ixMinute[] = {3,3,0,-3,-3}
        ,iyMinute[] = {0,-45,-50,-45,0};

    public void init()
    {
        this.setLayout(null);
        date = new Date();
        SimpleDateFormat df = new SimpleDateFormat("现在时刻' hh:mm:ss");
        String lasttime = df.format(date);
        offI = createImage(300,300);
        offG = offI.getGraphics();
        backG = getImage(getCodeBase(),"Clock.jpg");
    }

    public void start()
    {
        if(thread == null)
        {
            thread = new Thread(this);
            thread.start();
        }
    }

    public void run()
    {
        RotateG g = new RotateG(offG);
```

```

while(true)
{
    try
    {
        thread.sleep(1000);
        date = new Date();
        SimpleDateFormat df = new SimpleDateFormat("现在时刻' hh:mm:ss");
        String lasttime = df.format(date);
        System.out.println(lasttime);
        iSecond = date.getSeconds();
        iMinute = date.getMinutes();
        iHour = date.getHours();
        offG.setColor(Color.white);
        offG.fillRect(0,0,300,300);
        g.setOrigin(0,0);
        g.drawImage(backG,0,0,this);
        g.setOrigin(125,120);
        System.out.println(iHour);
        g.setAngle(iHour * 30 + iMinute * 0.5);
        g.setColor(Color.green);
        g.fillPolygon(ixHour, iyHour, 5);
        g.setAngle(iMinute * 6);
        g.setColor(Color.red);
        g.fillPolygon(ixMinute, iyMinute, 5);
        g.setAngle(iSecond * 6);
        g.setColor(Color.blue);
        g.drawLine(0,0,0,-65);
        g.setColor(Color.black);
        g.fillOval(120,115,10,10);
        repaint();
    }
    catch(InterruptedException e){}
}

public void update(Graphics g)
{
    paint(g);
}

public void paint(Graphics g)
{
    g.drawImage(offI,0,0,this);
}

```

```
}

//RotateG.java
import java.awt.*;
import java.awt.image.*;

public class RotateG
{
    private Graphics g;
    private int ox = 0;
    private int oy = 0;
    private double radians = 0.0;
    private double cos = 1.0;
    private double sin = 0.0;

    public RotateG(Graphics g)
    {
        this.g = g.create();
    }

    public void setOrigin(int x, int y)
    {
        ox = x;
        oy = y;
    }

    public void setAngle(double a)
    {
        radians = (a * 2 * Math.PI) / 360;
        cos = Math.cos(radians);
        sin = Math.sin(radians);
    }

    public void setColor(Color c)
    {
        g.setColor(c);
    }

    public int rotate_x(int x, int y)
    {
        return ((int) (ox + x * cos - y * sin));
    }

    public int rotate_y(int x, int y)
    {

```

```
        return ((int) (oy + y * cos + x * sin));
    }

    public void drawLine(int x1, int y1, int x2, int y2)
    {
        g.drawLine(rotate_x(x1, y1), rotate_y(x1, y1),
            rotate_x(x2, y2), rotate_y(x2, y2));
    }

    public void drawOval(int x, int y, int width, int height)
    {
        g.drawOval(x, y, width, height);
    }

    public void fillOval(int x, int y, int width, int height)
    {
        g.fillOval(x, y, width, height);
    }

    public void drawPolygon(int x[], int y[], int n)
    {
        int new_x[] = new int[n];
        int new_y[] = new int[n];

        for (int i = 0; i < n; i++)
        {
            new_x[i] = rotate_x(x[i], y[i]);
            new_y[i] = rotate_y(x[i], y[i]);
        }

        g.drawPolygon(new_x, new_y, n);
    }

    public void fillPolygon(int x[], int y[], int n)
    {
        int new_x[] = new int[n];
        int new_y[] = new int[n];

        for (int i = 0; i < n; i++)
        {
            new_x[i] = rotate_x(x[i], y[i]);
            new_y[i] = rotate_y(x[i], y[i]);
        }

        g.fillPolygon(new_x, new_y, n);
    }
```

```
public void drawImage(Image img,int x, int y,ImageObserver observ-
                        er)
{
    g.drawImage(img,x,y,observer);
}
}
```



更上一层楼

在本章我们学习了如何用 Visual J++ 创建 Java 小程序,了解了 Java 小程序的一些相关知识。

其实关于 Java 小程序还有许多需要学习的内容,这一章只是一个入门,关于 Java 小程序的进一步内容,例如 Java 小程序与 JavaScript 的配合使用、小程序与小程序之间的通信、小程序的界面元素的应用、布局管理、多线程应用、在小程序中使用多媒体等,需要读者自己去找相关的资料进行深入的了解。下一章我们将对其中某些深入的内容进行讨论。

第 7 章 Java 小程序编程进阶

知识要点：

- ◆ 运用小程序的界面组件进行动态网页设计
- ◆ 对 Web 页面上的小程序界面组件进行布局管理
- ◆ 创建多线程小程序

本章将在第 6 章的基础上进一步学习 Java 小程序的编程,我们在本章学习的主题将涉及到 Java 小程序中最常用的界面组件,用于界面设计的布局管理,以及 Java 小程序的高级应用——多线程小程序。



光盘 请参阅本书配套光盘的【创建小程序】和【小程序中的动画】可交互学习与本章内容相关的知识。

7.1 小程序的界面组件

在 Java 中,一个小应用程序的用户界面大多由以下组件组合而成:

- ◆ 按钮(Button)
- ◆ 文本框(TextField)
- ◆ 文本域(TextArea)
- ◆ 标签(Label)
- ◆ 复选框(Checkbox)
- ◆ 列表(List)
- ◆ 选项栏(Choice)

当然这些只是众多界面组件的一部分,这些界面组件可以按任意方式组合成小应用所需的用户界面。我们先来看一个例子,如图 7.1 所示(随书光盘的 Samples\chap07\ex07a 下)。



图 7.1 小程序的界面组件示例

在图 7.1 所示的小程序界面中,有 6 种界面组件,如表 7.1 所示。

表 7.1 界面组件示例

组件示例	说明
你的姓名:	标签(Label)
<input type="text" value="刘杰"/>	文本框(TextField)
<input checked="" type="radio"/> 男 <input type="radio"/> 女	分组的复选框(Checkbox)
<input checked="" type="checkbox"/> Java <input type="checkbox"/> C++	复选框(Checkbox)
计算机世界	选项栏(Choice)
我认为	文本域(TextArea)
重写	按钮(Button)

这些组件其实都是定义在 `java.awt` 包中的类,只要创建相应类的对象就可以在小程序中生成这些界面组件。本节我们将对按钮、文本框、文本域和复选框等几个界面组件的用法作详细介绍,其他更多的界面组件由于篇幅限制,在此就不介绍了。不过通过本节的学习您可以举一反三,迅速了解其他界面组件的用法。详细信息请在 Visual J++ 的帮助中查阅关于 Package `java.awt` 的内容,在帮助中您可以查到每种组件的详细用法。

我们刚才介绍的界面组件大多是从 `Component` 类继承下来的,也就是说这些界面组件都具有 `Component` 类的属性、事件和方法。我们要了解界面组件,就需要首先了解一下 `Component` 类。`Component` 类也是在 `java.awt` 包中定义的。由于它可以把事件从 `Component` 的一个子类方便地传递给另一个子类,因此 `Component` 是绝大多数 Java 用户界面类的父类。作为 Java 用户界面的父类, `Component` 类并不只是为每个用户界面对象提供一级共同的事件处理方法。`Component` 是一个很大的类,共有 70 多个方法,其子类都可以使用这些已经定义好的方法。其中最常用方法在表 7.2 中列出。

表 7.2 `Component` 中最常用的方法

方法	作用
Action	操作事件发生时,所调用的事件处理方法
Disable	禁用组件
Enable	启用组件

续表

方法	作用
getBackground	获得组件背景
GetFont	获得组件字体
getGraphics	获得组件的图形对象的参考索引
GetParent	获得组件的父对象
GetFocus	组件得到焦点时调用的事件处理方法
handleEvent	通用的事件处理方法
Hide	隐藏组件
Inside	确定给定坐标(x,y)是否在组件内
isEnabled	如果组件为启用状态,则返回真
isVible	如果组件可见,则返回真
LostFocus	组件失去焦点时调用的事件处理方法
minimumSize	返回组件所需的最小尺寸
mouseDown	事件处理方法,鼠标按下时调用
mouseUp	事件处理方法,鼠标释放时调用
mouseDrag	事件处理方法,鼠标按住并移动时调用
mouseEnter	事件处理方法,鼠标进入组件时调用
mouseExit	事件处理方法,鼠标离开组件时调用
mouseMove	事件处理方法,鼠标移动且按键松开时调用
move	移动组件
nextFocus	把焦点给下一个组件
paint	绘制组件
preferredSize	返回组件的推荐尺寸
repaint	重绘组件
resize	改变组件的尺寸
setBackground	设置组件的背景色
setFont	设置组件所用字体
setForeground	设置组件的前景色
show	使组件可见

详细信息请在 Visual J++ 的帮助中查阅关于 Component Members 的内容,在帮助中可以查到 Component 类的所有成员变量和方法。

下面我们通过几个使用界面组件的例子来学习如何使用它们。

7.1.1 按钮(Button)

按钮是最简单的 Java 用户界面类之一。创建按钮可以使用表 7.3 中所示的构造函数之一。例如,下面的语句就会创建一个标签为“重写”的按钮。

```
Button myButton;  
myButton = new Button("重写");
```

在本节一开始的小程序的界面组件示例中的按钮就是这样创建的,参看图 7.1。构造函数 `Button()` 和 `Button(String label)` 的不同之处在于使用前者创建的按钮没有标签,而使用后者创建的按钮有标签,显然,采用标签作参数的构造函数要比没有参数的构造函数实用。

表 7.3 按钮构造函数

构造函数	作用
<code>Button()</code>	创建一个无标签的按钮
<code>Button(String label)</code>	创建一个有标签的按钮

创建并显示按钮

下面我们一起做练习 ex07b(随书光盘的 \Samples\chap07\ex07b 下)。这个练习很简单,它创建一个按钮并放在小应用程序中。

把创建好的按钮放到 Applet 上,是通过调用 `add()` 方法来实现的。如果不调用 `add()` 方法,那么将只是创建按钮,我们却看不到它。由于 `add()` 方法在 `java.awt.Container` 中定义,而 Applet 也是 `Container` 的子类,所以在程序中可以直接调用 `add()` 方法。程序运行结果如图 7.2 所示。



图 7.2 创建并显示按钮

程序代码:



```
import java.applet.*;
import java.awt.*;
public class Applet1 extends Applet
{
    Button myButton;
    public void init()
    {
        myButton = new Button("push me");
        add(myButton);
    }
}
```

处理按钮事件

在第6章中我们已学习过在小程序中如何处理鼠标事件。尽管按按钮也是用户单击鼠标的结果,但 Java 还是提供了用于按钮的消息处理方法。现在我们来学习这种新的处理鼠标事件的方法——action()方法。

action()方法用于处理 Applet 上发生的各种事件,如按下按钮、选取复选框或选择菜单项等。action()方法以事件(Event)和事件的对象(Object)作为参数,它的声明如下:

```
public boolean action (Event evt, Object obj)
```

例如鼠标单击按钮 Btn,按钮 Btn 被按下,这时 action()方法的两个参数即为 MOUSE_DOWN 和 Btn。

因为 action()方法“监视”Applet 上发生的各种事件,因此 action()方法必须得知道事件是在 Applet 的哪个组件上发生的。那么 action()方法是如何知道它所接到的事件是从谁那里发出的呢?是通过 equals()方法知道的。

equals()方法是 Object 类的方法,它被 Object 类的对象调用,同时它又有一个 Object 类的参数,它的声明如下:

```
public boolean equals(Object)
```

Object 类的 equals()方法用于比较两个对象(调用 equals()方法和 equals()方法的参数对象)是否相同。例如,x,y 是 Object 类的两个对象,x.equals(y)如果为真(true),那么 x 就等于 y,即 x 和 y 实际上是同一个对象;如果为假,那么 x 和 y 就是不同的对象。

在我们的这个例子中,equals()方法用于比较 action()方法的 Object 参数 obj 和按钮的标签“push me”,如果二者相等,那么就可以判定发生事件的对象即为

按钮。我们还是通过一个例子来看究竟是怎么回事。

我们在前一个例子的基础上增加一个功能：当按钮被按下时，在 Applet 上显示一串字符。运行后的结果如图 7.3 所示，按下按钮后，在 Applet 上显示“you pushed me!”。通过这个例子我们就会了解鼠标事件是如何被处理的了。

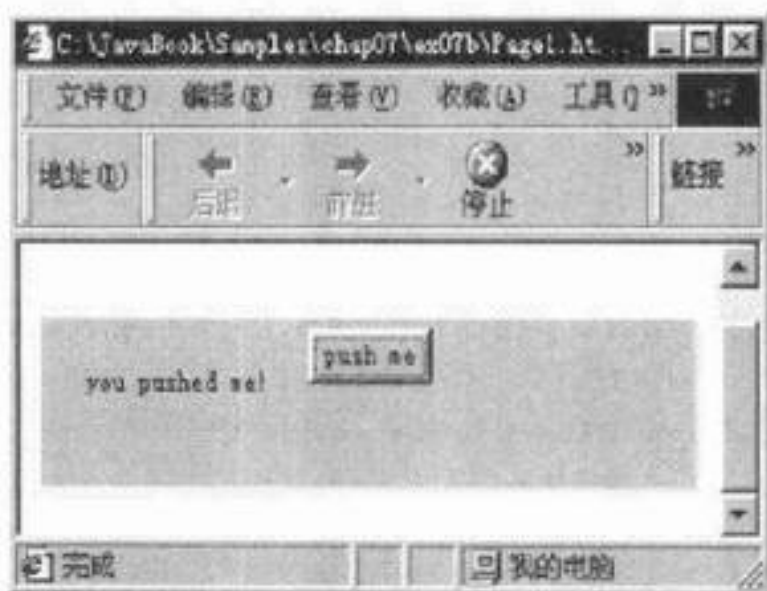


图 7.3 处理按钮事件

程序代码：



```
import java.applet.*;
import java.awt.*;
public class Applet1 extends Applet
{
    Button myButton;
    public void init()
    {
        myButton = new Button("push me");
        add(myButton);
    }
    public boolean action (Event evt, Object obj)
    {
        boolean result = false;
        if("push me".equals(obj))
        {
            getGraphics().drawString("you pushed me!", 20, 20);
            result = true;
        }
    }
}
```

```
return result;
```

在上面程序中关键是 `action (Event evt, Object obj)` 方法中的几条语句:



```
if("push me".equals(obj))
{
    getGraphics().drawString("you pushed me!",20,20);
    result = true;
}
```

我们来分析一下这几条语句。

首先,“push me”.equals(obj)的含义是,通过按钮 myButton 的标签“push me”来引用 myButton 对象(Java 语言支持这种简便的语法),这样“push me”就作为对象来调用 equals()方法。参数 obj 是 action()方法“捕捉”到的 Applet 上发生事件 evt 的对象,这样通过 equals()方法来比较发生了事件 evt 的对象 obj 和 myButton 是否是同一个对象。如果是,就说明事件 evt 是在 myButton 上发生的。另外,根据 equals()方法的语法,我们可以知道“push me”.equals(obj)其实等同于语句 obj.equals (“push me”)。

其次,经过比较判断,如果“push me”.equals(obj)为真,那么 action()方法就知道事件是按钮 myButton 发出的,于是利用“getGraphics().drawString(“you pushed me!”, 20,20);”语句在 Applet(20,20)处显示字符串“you pushed me!”。getGraphics()方法是 Component 类的方法, getGraphics()的作用是返回 component 类(这里是对象 Applet)的图像环境(graphics context),返回类型是 Graphics 类。由于 Applet 是 Component 类的子类,所以在程序中可以直接调用 getGraphics()方法来获得 Graphics 对象的引用。这样有了 Graphics 对象,就可以调用 Graphics 类的绘图方法,这里调用的绘图方法是 drawString(),以便在 Applet1 上显示字符串。

最后,通过语句“result = true;”返回 action()方法的结果值 true,表示事件处理完毕。如果 action()方法的结果值为 false,则表示事件没有被处理。

公用的按钮方法

除了构造函数和作为 Component 子类可使用的方法之外,Button 类还提供了表 7.4 中所列的公用成员方法。

表 7.4 Button 类的公用成员方法

方法	作用
<code>addNotify()</code>	创建一个同类组件
<code>getLabel()</code>	返回按钮的当前标签
<code> paramString()</code>	返回按钮的参数字符串
<code>setLabel(String)</code>	将按钮标签设为指定的字符串

下面的程序代码是练习 ex07c, 这是一个稍微复杂的例子, 演示了 Button 类的用法, 其中用到了 `setLabel(String)` 方法。在 ex07c 中我们创建两个按钮, 其中只有左边的按钮被单击两次后, 再单击右边的按钮才会显示一串字符, 如果不是按这个顺序, 那么字符串就不会出现。另外, 单击左边的按钮还会改变右边按钮的标签。程序中用 `if...else...` 语句来判断哪个按钮被单击。运行结果如图 7.4(a)、7.4(b)、7.4(c) 和 7.4(d) 所示。

程序代码:



```
import java.applet.*;
import java.awt.*;
public class Applet1 extends Applet
{
    int count = 0;
    Button button2 = new Button(" 计数器 ");
    public void init()
    {
        add(new Button("请按这个按钮两次!"));
        add(button2);
    }

    public boolean action(Event evt, Object obj)
    {
        boolean result = false;
        if("请按这个按钮两次!".equals(obj)) {
            count++;
            switch(count)
            {
                case 1:
                    button2.setLabel("一次");
                    break;
                case 2:
                    button2.setLabel("两次,OK!");
                    break;
            }
        }
    }
}
```

```
        result = true;
    }
    else if ("两次,OK!".equals(obj)){
        getGraphics().drawString("这是我的存折密码-7788",20,60);
        result = true;
    }
    return result;
}
}
```



图 7.4(a) ex07c 的开始界面



图 7.4(b) 单击左边按钮一次后



图 7.4(c) 单击左边按钮两次后

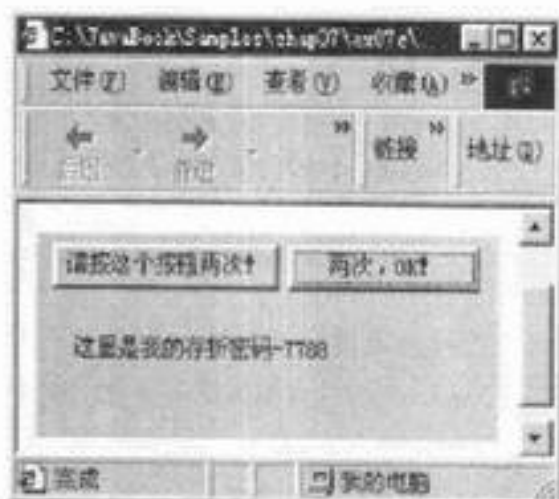


图 7.4(d) 单击右边按钮后

当 ex07c 开始运行时,右边按钮的标签为“计数器”,按下左边的按钮将激活小应用程序的 `action()` 方法,它记录按钮被按的次数。当左边按钮被单击一次时,右边按钮的标签变为“一次”。第二次单击左边按钮后,右边按钮的标签变为“两次, OK!”。这时单击右边按钮,才会显示一串字符“这是我的存折密码-7788”。

7.1.2 文本框(TextField)和文本域(TextArea)

创建文本框、文本域

文本框是一个单行的数据输入栏,文本域则是包含滚动条的多行文本框。

创建文本框或文本域的方法有多种,例如:

```
TextField tf1 = new TextField(25)
TextField tf2 = new TextField("This is a Test Field")
TextArea ta1 = new TextArea(10,50)
TextArea ta2 = new TextArea("This is a 10 x 50 Test Area", 10,50);
```

其中,tf1 是一个能容纳 25 个字符的文本框,tf2 则是一个显示字符串“This is a Test Field”的文本框,它的宽度正好容下这个字符串。两个文本域 ta1 和 ta2 都是高 10 字符、宽 50 字符大小,但 ta2 包含有显示出来的初始字符串“This is a 10 x 50 Test Area”。

文本框是由 TextField 类的构造函数创建的,TextField 类的构造方法有 4 种,在表 7.5 中列出。

表 7.5 TextField 类的构造函数

构造函数	作用
TextField()	创建一个空的单行文本输入框
TextField(int)	创建一个指定长度的单行文本输入框
TextField(String)	创建一个指定初始内容的单行文本输入框
TextField(String, int)	创建一个指定长度、指定初始内容的单行文本输入框

在某种情况下,用户可能希望自己的输入不被别人看到,这时可以用 TextField 类中的 setEchoCharacter()方法设置回显字符,使用户的输入全部以某个特殊字符显示在屏幕上。

例如,执行下面的语句会创建一个名为 Password 的文本框,但在里面输入内容时,这个文本框中都显示“*”。

```
TextField Password = new TextField(10);
Password.setEchoCharacter('*');
add(Password);
```

结果形如: 。

文本域其实就是多行的文本框,功能与文本框的功能差不多,只是它能显示更多的文字。因为文本框只能输入一行文字,所以在需要输入和显示较多的文字时,就要用到文本域。文本域是由 TextArea 类的构造函数创建的,TextArea 类的构造方法有 4 种,在表 7.6 中列出。

表 7.6 TextArea 类的构造函数

构造函数	作用
TextArea ()	创建一个空的文本域
TextArea (int,int)	创建一个给定行数、列数的文本域
TextArea (String)	创建一个含有给定字符串的文本域
TextArea (String, int, int)	创建一个行数、列数给定的,含有给定字符串的文本域

公用的文本框、文本域方法

TextField 和 TextArea 都是 TextComponent 类的子类(TextComponent 类是 Component 类的子类)。TextComponent 类所提供的公用成员方法如表 7.7 中所示。

表 7.7 TextComponent 的公用成员方法

方法	作用
getSelectedText ()	返回当前选中的字符串
getSelectionEnd ()	返回被选字符串的结尾列号
getSelectionStart ()	返回被选字符串的开头列号
getText ()	返回 TextComponent 中的文本
isEditable ()	如果 TextComponent 可编辑则返回 true
paramString ()	返回 TextComponent 的参数字符串
RemoveNotify ()	删除 TextComponent 的同类组件
select (int,int)	选定所给列之间的文本
selectAll ()	选定所有文本
setEditable (boolean)	指定 TextComponent 是否可由用户编辑
setText (String)	设置 TextComponent 包含指定字符串

除了可共同分享的 TextComponent 类的方法外,TextField 和 TextArea 还有各自的成员方法,如表 7.8 和 7.9 中所列。

表 7.8 TextField 的成员方法

方法	作用
AddNotify ()	创建一个同类组件
echoCharIsSet ()	如果设置了响应字符,则返回 true
getColumns ()	返回的列数
getEchoChar ()	返回将使用的响应字符
minimumSize (int)	返回具有给定列数的 TextField 的最小尺寸

续表

方法	作用
<code>paramString()</code>	返回 <code>TextFied</code> 的参数字符串
<code>preferredSize(int)</code>	返回列数给定的 <code>TextFied</code> 的推荐尺寸
<code>preferredSize()</code>	返回 <code>TextFied</code> 的推荐尺寸
<code>getEchoCharacter(char)</code>	将回应字符设为给定字符

表 7.9 `TextArea` 的成员方法

方法	作用
<code>AddNotify()</code>	创建一个同类组件
<code>appendText()</code>	将给定字符串附加到 <code>TextArea</code> 末尾
<code>getColumns()</code>	返回 <code>TextArea</code> 的列数
<code>getRows()</code>	返回 <code>TextArea</code> 的行数
<code>insertText(String,int)</code>	在指定列处插入给定字符串
<code>minimumSize(int,int)</code>	返回行、列数给定的 <code>TextArea</code> 的最小尺寸
<code>minimumSize()</code>	返回 <code>TextArea</code> 的最小尺寸
<code>paramString()</code>	返回 <code>TextArea</code> 的参数字符串
<code>preferredSize(int,int)</code>	返回行、列数给定的 <code>TextArea</code> 的推荐尺寸
<code>preferredSize()</code>	返回 <code>TextArea</code> 的推荐尺寸
<code>ReplaceText(String,int,int)</code>	用指定字符串代替给定列之间的文本

文本框、文本域示例

下面的程序代码是练习 ex07d 的源代码,练习 ex07d 演示了文本框和文本域的用法。练习中创建了一个文本框 `tf` 和一个文本域 `ta`。在 `init()` 方法中,还创建了一个标签为“添加”的按钮,并且每个组件都被加到小应用程序中。运行结果如图 7.5 所示。

当“添加”按钮被按下时,文本框 `tf` 中的文本将被复制到文本域 `ta`,添加到已有内容的后面,同时文本框 `tf` 被清空。

程序代码:



```
import java.applet.*;
import java.awt.*;
public class Applet1 extends Applet
{
    TextField tf= new TextField(20);
    TextArea ta= new TextArea(" 一球星花名册→\r\n",6,25);
    public void init()
```

```

    }
    add(tf);
    add(ta);
    add(new Button("添加"));
}
public boolean action(Event evt, Object obj)
{
    boolean result = false;
    if("添加".equals(obj)){
        ta.appendText(tf.getText() + "\r\n");
        tf.setText("");
        result = true;
        return result;
    }
    else return result;
}
}

```



图 7.5 练习 ex07d 的运行结果

ex07d 中的 `action()` 方法查看是否有“添加”按钮产生的操作。如果有, 则使用 `tf.getText()` 方法和 `ta.appendText()` 方法把文本框 `tf` 中的文本复制到文本域 `ta` 的末尾。文本框 `tf` 中的文本被复制后利用 `tf.setText("")` 语句被置为空串。

7.1.3 复选框(Checkbox)和复选框组(CheckboxGroup)

复选框和复选框组用来让用户选择某些选项。Java 的复选框有两种类型:

成组的和不成组的。Java 的复选框用 `CheckboxGroup` 类分组。如果一个 `CheckboxGroup` 中有一个以上的复选框,则任何时刻这些复选框中都只能有一个被选中。在其他编程环境中,成组的复选框习惯上称作单选按钮(`RadioButton`)。在本章一开始的例子中即有成组和不成组的复选框的实例(参看图 7.6)。成组的复选框只能单选,“男”和“女”复选框分在同一组中,因而互相排斥。不成组的复选框被选或不被选则无需考虑其他复选框,例如 Java、C++ 等各种“编程语言”选项联合成为一个不成组的复选框,这样可以进行多项选择,因为许多情况下用户可能需要一个以上的选项。

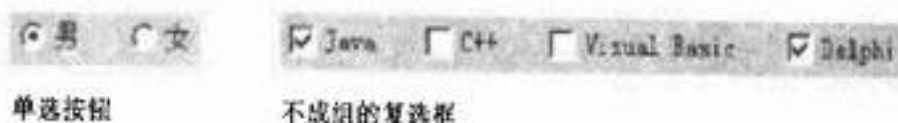


图 7.6 复选框对比

创建复选框和复选框组

`Checkbox` 类有 3 个构造函数,如表 7.10 所示。

表 7.10 `Checkbox` 类的构造函数

构造函数	作用
<code>Checkbox()</code>	创建无标签复选框
<code>Checkbox(String)</code>	创建以 <code>String</code> 为标签的复选框
<code>Checkbox(String, CheckboxGroup, boolean)</code>	创建属于指定组、以 <code>String</code> 为标签并设置为默认值的复选框

这里我们讲解上面第 3 种构造函数——`Checkbox(String, CheckboxGroup, boolean)`。它的 `String` 参数设定复选框的标签。`CheckboxGroup` 参数设定复选框的组,如果 `CheckboxGroup` 参数为 `null`,那么该复选框不成组,可以多选,如果 `CheckboxGroup` 参数不为 `null`,则该复选框属于 `CheckboxGroup` 组,只能单选。`boolean` 参数设定复选框的初始状态为选中还是未选中,`true` 是选中,`false` 是未选中。

在下面的代码中,创建了一个与图 7.6 相同的复选框组,以及 4 个不成组的复选框。



```
CheckboxGroup genderGroup = new CheckboxGroup();
add(new Checkbox("男", genderGroup, true));
add(new Checkbox("女", genderGroup, false));
```

```

add (new Checkbox("Java"));
add (new Checkbox("C++"));
add (new Checkbox("Visual Basic"));
add (new Checkbox("Delphi"));

```

复选框、复选框组的成员方法

除构造函数外,Checkbox 和 CheckboxGroup 类还提供了其他成员方法,见表 7.11 和 7.12。

表 7.11 Checkbox 类的成员方法

方法	作用
addNotify ()	创建一个同类组件
getCheckboxGroup ()	返回复选框所属的复选框组
getLabel ()	返回复选框的标签
getState ()	返回复选框的状态,如选中则返回 true
paramString ()	返回参数字符串
setCheckboxGroup (CheckboxGroup)	将复选框设为属于指定组
setLabel (String)	设定复选框的标签
setState (Boolean)	复选框的状态

表 7.12 CheckboxGroup 类的成员方法

方法	作用
getSelectedCheckbox ()	返回当前选中的复选框
setSelectedCheckbox (Checkbox)	将指定复选框设为组中当前的选择
toString ()	返回一个表示 CheckboxGroup 当前值的字符串

复选框、复选框组示例

下面的程序代码是练习 ex07e 的源代码,练习 ex07e 演示了复选框和复选框组的用法。练习中通过 init()方法添加了两个复选框到性别 CheckboxGroup,添加了 5 个不成组的复选框,一个【查看】按钮和一个文本域。

程序运行结果如图 7.7 所示。单击【查看】按钮时,每个被选中复选框的标签都写到文本域中。如果重新对复选框进行选择,再次单击【查看】按钮时,文本域将清除前一次选择结果,反映当前所做的选择。



图 7.7 练习 ex07c 的运行结果

程序代码:



```
import java.applet.*;
import java.awt.*;
public class Applet1 extends Applet
{
    CheckboxGroup genderGroup = new CheckboxGroup();
    Checkbox Checkbox1 = new Checkbox("篮球");
    Checkbox Checkbox2 = new Checkbox("足球");
    Checkbox Checkbox3 = new Checkbox("游泳");
    Checkbox Checkbox4 = new Checkbox("溜冰");
    Checkbox Checkbox5 = new Checkbox("跑步");
    TextArea results = new TextArea(4,10);

    public void init()
    {
        add(new Label("性别:"));
        add(new Checkbox("男", genderGroup, false));
        add(new Checkbox("女", genderGroup, true));
        add(new Label("你喜欢什么体育活动?"));
        add(Checkbox1);
        add(Checkbox2);
        add(Checkbox3);
        add(Checkbox4);
        add(Checkbox5);
        add(new Button("查看"));
        add(results);
    }
}
```

```

public boolean action(Event evt, Object obj)
{
    boolean result = false;
    if("查看".equals(obj))
    {
        // clear the results area
        results.setText("");
        // display the gender
        Checkbox current = genderGroup.getSelectedCheckbox();
        results.appendText(current.getLabel() + "\r\n");
        // check each of the sports
        if (Checkbox1.getState() == true)
            results.appendText(Checkbox1.getLabel() + "\r\n");
        if (Checkbox2.getState() == true)
            results.appendText(Checkbox2.getLabel() + "\r\n");
        if (Checkbox3.getState() == true)
            results.appendText(Checkbox3.getLabel() + "\r\n");
        if (Checkbox4.getState() == true)
            results.appendText(Checkbox4.getLabel() + "\r\n");
        if (Checkbox5.getState() == true)
            results.appendText(Checkbox5.getLabel() + "\r\n");
        result = true;
    }
    return result;
}

```

像本节中的其他例子一样, `action()` 方法负责检查【查看】按钮是否按下。在本例中按下按钮时文本域 `results` 被清除。接着通过语句

```
Checkbox current = genderGroup.getSelectedCheckbox();
```

获得 `genderGroup` 中被选的复选框, 并将被选的复选框赋值给 `Checkbox` 类对象 `current`。再通过语句

```
results.appendText(current.getLabel() + "\r\n");
```

将 `genderGroup` 中被选的复选框的标签写到 `results` 中。

本节介绍了用于构建小程序用户界面的界面组件, 主要介绍了按钮 (`Button`)、文本框 (`TextField`)、文本域 (`TextArea`)、复选框 (`Checkbox`) 和复选框组 (`CheckboxGroup`)。我们学会了如何通过界面组件来设计小程序用户界面, 但是正如您所看到的, 这些界面组件通过 `add()` 方法被自动放在 `Applet` 的某个地方, 您却改变不了它们的位置, 所以界面看起来有些杂乱。下一节我们将学习如何控制界面组件在 `Applet` 上的摆放, 这就是布局管理。

7.2 布局管理

我们使用 `add()` 方法将用户界面组件放在 Applet 上时,不必告诉 Java 在哪里放置每个组件,Java 知道每个组件属于何处,并将它放在那里。这其实是 Java 利用布局管理器来放置每个组件。但从结果来看,如果只是简单地把组件添加到 Applet 上,效果往往不大理想,所以必须进行版面设计,这一节我们将介绍如何利用布局管理进行排版。

7.2.1 布局管理器

在以往的用户界面中,每个组件的位置一般是用点坐标确定的。但是在 Java 中,情况有所不同。Java 的 Applet 需要运行在不同操作系统、不同的屏幕设置、不同的字体下,用点坐标来确定位置就难以适应各种情况,所以在 Java 中使用布局管理器来进行排版。使用布局管理器的优点是能根据不同的屏幕自动进行排版,缺点是组件在屏幕上的确切位置难以确定。组件在屏幕上的确切位置只取决于当前布局管理器所使用的版面,以及组件添加到屏幕上的顺序。

Java 的软件开发工具箱(SDK)支持多个布局管理器,FlowLayout(流布局管理器)、BorderLayout(边框布局管理器)、GridLayout(网格布局管理器)和 GridBagLayout(网袋布局管理器)等。您还可以创建自己的布局管理器。包含用户界面元件的 Java 类,如 Applet、Panel、Frame 和 Dialog 类,都可以使用布局管理器。每个类有一个默认的布局管理器(如表 7.13 所示),但这些管理器可以被这几个类中的任何一个使用。

表 7.13 默认的布局管理器

类	默认的布局管理器
Applet	FlowLayout
Dialog	BorderLayout
Frame	BorderLayout
Panel	FlowLayout

在 Applet 中您可以使用下面的语句来选择布局。

```
public void init() {
    setLayout( new BorderLayout); //这里我们选择的是边框布局
}
```

7.2.2 FlowLayout(流布局管理器)

FlowLayout 是 Applet 类默认的布局管理器,所以本章到此为止的所有例子都默认地用到了它。FlowLayout 所做的就是把组件从左向右连续不断地放到同一行中,直到放不下为止。如果一个组件在当前行放不下,FlowLayout 就把组件移到下一行。组件按其被加入到小程序的顺序在屏幕上出现。当有大量组件要用时,由于组件尺寸的关系,FlowLayout 难以达到预定的效果,组件并不能如希望的那样排列。但在组件为数不多时,FlowLayout 还是一个不错的选择。

在 FlowLayout 中也可设置版面的对齐方式,可以是居中、靠左或靠右,默认是居中。要设置对齐方式,可以使用 FlowLayout 类中的变量 LEFT、CENTER 和 RIGHT,语句如下:

```
setLayout (new FlowLayout(FlowLayout.LEFT));
```

此外 FlowLayout 中还可以设置横向和纵向的间隔(默认是 3 个像素)。设置间隔可以使用下面的语句:

```
setLayout (new FlowLayout(FlowLayout.CENTER,1,5));
```

语句执行后将横向间隔设置成 1 个像素,把纵向间隔设置成 5 个像素。

我们创建 5 个按钮,利用 FlowLayout 进行左对齐布局,看看效果如何。图 7.8 是它的版面效果。



```
import java.awt.*;
import java.applet.*;

public class Applet1 extends Applet
{
    public void init()
    {
        setLayout (new FlowLayout(FlowLayout.LEFT));
        add(new Button("1"));
        add(new Button("2"));
        add(new Button("3"));
        add(new Button("4"));
        add(new Button("5"));
    }
}
```



图 7.8 FlowLayout 左对齐布局

我们再来看看设置间隔的效果,利用 FlowLayout 对 5 个按钮进行中对齐布局,将横向间隔设置成 1 个像素,把纵向间隔设置成 5 个像素。使用前面的语句:

```
setLayout (new FlowLayout(FlowLayout.CENTER,1,5));
```

执行后的版面如图 7.9。



图 7.9 FlowLayout 设置间隔、中对齐布局

7.2.3 BorderLayout(边框布局管理器)

当要放置的组件数量不太多,而且又想对它们进行比 FlowLayout 更为严格的控制时,BorderLayout 就非常有用。BorderLayout 最多可控制 5 个组件,每个组件都放在下列区域之一:

- ◆ 北(North)
- ◆ 南(South)
- ◆ 东(East)
- ◆ 西(West)
- ◆ 中(Center)

和习惯一样, North 是指对象的顶部, South 则指底部, East 为右边, West 为左边, Center 为中。通常放在 Center(中)位置上的组件要大一些,例如,下面的示例演示了 5 个按钮,各占一个位置。请注意 Center 位置上的按钮,它自动放大填满可用的空间(如图 7.10)。



```
setLayout (new BorderLayout());
add("North", new Button("1"));
add("South", new Button("2"));
add("West", new Button("3"));
```



```
add("East", new Button("4"));
add("Center", new Button("5"));
```

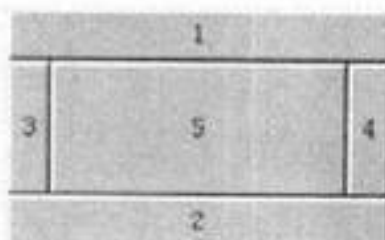


图 7.10 BorderLayout 布局

BorderLayout 中也可设置间隔,效果如图 7.11 所示。设置方法如下:

```
setLayout (new BorderLayout(1,5));
```

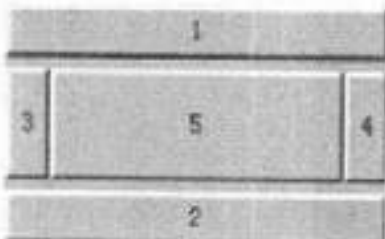


图 7.11 BorderLayout 布局设置间隔

当使用 BorderLayout 加入组件时,可以指定组件所放的位置。例如,下列代码将把一个按钮放到 North(北),另一按钮放到 South(南),效果如图 7.12 所示。

```
add ("North",new Button("OK"));
add ("South",new Button("Cancel"));
```



图 7.12 BorderLayout 布局设置两个按钮位置

7.2.4 GridLayout(网格布局管理器)

Grid 在英文中是格子的意思,在 GridLayout 中 Applet 被划分成网格状,每一个组件按照添加的顺序从左向右、从上向下地占据一个单元。所以在 GridLayout

中,组件添加的顺序相当重要。GridLayout 在有一组大小相同的控制要布置时很有用。

要创建一个 2×3 的网格版面可以用以下的语句,图 7.13 是执行的结果。

```
setLayout(new GridLayout(2,3));
```



图 7.13 GridLayout 布局

同样,GridLayout 中也可以设置间隔。下面的语句把横向间隔设置成 5 个像素,把纵向间隔设置成 10 个像素(如图 7.14)。

```
setLayout( new GridLayout(2, 3, 5, 10);
```



图 7.14 GridLayout 布局设置间隔

下面我们看一个利用 GridLayout 放置不同界面组件的例子,结果如图 7.15 所示。



```
import java.applet.*;
import java.awt.*;
public class Applet1 extends Applet
{
    public void init()
    {
        setLayout ( new GridLayout(3,3,2,2) );
        add (new Label(" 标签"));
        add (new Label(""));
        add (new Label(""));
        add (new Label(""));
        add (new TextField(" 文本框"));
    }
}
```

```

add (new Label(""));
add (new Checkbox("复选框"));
add (new Label(""));
add (new Button("按钮"));

```

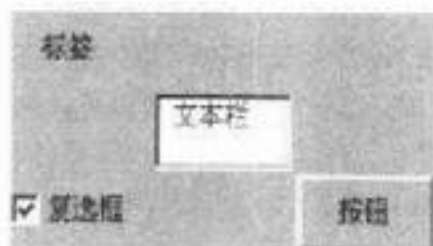


图 7.15 GridLayout 布局布置不同界面组件

7.2.5 GridBagLayout(网袋布局管理器)

GridBagLayout 是一种非常灵活的版面布局,提供了布置组件的极大灵活性。它是在将屏幕划分成网格的基础上,允许每个组件占据一个或多个单元(显示区域)。GridBagLayout 管理器的每个组件都有一个相应的 GridBagConstraints 实例,通过这个实例来安排组件的位置。要有效地使用 GridBagLayout 类,必须为该组件定义一个 GridBagConstraints 实例,并正确设置实例变量。

实例变量如下:

- ◆ gridx 和 gridy 设置安放组件的网格单元的坐标。屏幕左上角的网格单元坐标是 gridx = 0, gridy = 0。如果使用系统默认值 GridBagConstraints.RELATIVE,那么这个组件将放置在前一个添加的组件的右边或下边。
- ◆ gridwidth 和 gridheight 以网格单元为单位设置显示区域的宽度和高度,默认值为 1。使用 GridBagConstraints.REMAINDER 可以设置这个组件是这一行或这一列中的最后一个,即占据剩下的网格单元。使用 GridBagConstraints.RELATIVE将这个组件设置成占据这一行或这一列中除最后一个以外的所有网格单元。
- ◆ fill 当显示区域大于组件实际尺寸时,设置如何重新安排组件的大小。可以使用的值有 GridBagConstraints.NONE(默认值)、GridBagConstraints.HORIZONTAL(将组件横向扩充以填满显示区域)、GridBagConstraints.VERTICAL(将组件纵向扩充以填满显示区域)和 GridBagConstraints.BOTH(将组件扩充以填满显示区域)。

- ◆ `ipadx` 和 `ipady` 设置组件之间的间隔。组件间的横向间隔为 `ipadx * 2` 个像素, 同样纵向间隔为 `ipady * 2` 个像素。
- ◆ `insets` 设置组件与屏幕边缘之间的间隔。
- ◆ `anchor` 当组件的尺寸比显示区域小时, 用来设置放置组件的位置(可以和 `fill` 参数一起使用), 其值取 `GridBagConstraints.CENTER` (`NORTH`、`SOUTH`、`EAST`、`WEST`、`NORTHEAST`、`NORTHWEST`、`SOUTHEAST`、`SOUTHWEST`)。
- ◆ `weightx` 和 `weighty` 用来设置如何分配空白区域(用于屏幕大小改变时), 其默认值是 0。所有的组件集中在容器(container)的中央。所以需要改变设置时, 必须给这两个参数赋值。

下面来看一个例子, 运行结果如图 7.16 所示。



```
import java.applet.*;
import java.awt.*;

public class Applet1 extends Applet
{
    public void init()
    {
        GridBagLayout layout = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();
        setLayout(layout);
        //add the first button
        gbc.fill = GridBagConstraints.BOTH;
        Button button = new Button("左上");
        layout.setConstraints(button, gbc);
        add(button);
        //add a button to the same row
        gbc.gridwidth = GridBagConstraints.RELATIVE;
        button = new Button("中上");
        layout.setConstraints(button, gbc);
        add(button);
        //place a final button on the first row
        gbc.gridwidth = GridBagConstraints.REMAINDER;
        button = new Button("右上");
        layout.setConstraints(button, gbc);
        add(button);
        //create a button the width of the applet
        gbc.weighty = 1.0;
        button = new Button("满行");
        layout.setConstraints(button, gbc);
        add(button);
        //create a button in the left of the bottom
        gbc.gridwidth = GridBagConstraints.RELATIVE;
        button = new Button("左下");
        layout.setConstraints(button, gbc);
    }
}
```

```

        add(button);
        //create a button in the right of the bottom
        gbc.gridwidth = GridBagConstraints.RELATIVE;
        button = new Button("右下");
        Layout.setConstraints(button,gbc);
        add(button);
    }
}

```



图 7.16 GridBagLayout 布局

7.2.6 综合使用

在介绍了这些布局之后,如何才能安排出令人满意的效果呢?我们可以把一系列的面板(Panel)添加到 Applet 上,在每个面板内部可以设置成不同的布局,添加不同的组件,这样就可以得到较好的效果。现在,我们来看一个综合使用的例子,运行结果如图 7.17 所示。您可以对照本节一开始的例子,对比使用布局管理与不使用布局管理的区别。



```

import java.awt.*;
import java.applet.*;

public class Applet1 extends Applet {
    public void init() {
        setLayout(new FlowLayout(FlowLayout.LEFT));
        Panel p1 = new Panel();
        add(p1);
        p1.add(new Label("你的姓名:"));
        p1.add(new TextField(30));
        Panel p2 = new Panel();
        add(p2);
        p2.add(new Label("性别:"));
        CheckboxGroup cbg = new CheckboxGroup();
        p2.add(new Checkbox("男", cbg, true));
        p2.add(new Checkbox("女", cbg, false));
        Panel p3 = new Panel();
    }
}

```

```

add(p3);
p3.add(new Label("你喜欢什么体育活动?"));
p3.add(new Checkbox("篮球"));
p3.add(new Checkbox("足球"));
p3.add(new Checkbox("游泳"));
p3.add(new Checkbox("溜冰"));
Panel p4 = new Panel();
add(p4);
p4.add(new Label("你最喜欢哪份报纸?"));
Choice c = new Choice();
c.addItem("计算机世界");
c.addItem("中国计算机报");
c.addItem("网络世界");
p4.add(c);
Panel p5 = new Panel();
add(p5);
p5.add(new Label("你对本书有何建议?"));
add(new TextArea("我认为",3,60));
Panel p6 = new Panel();
add(p6);
p6.add(new Button("提交"));
p6.add(new Button("重写"));
}
}

```

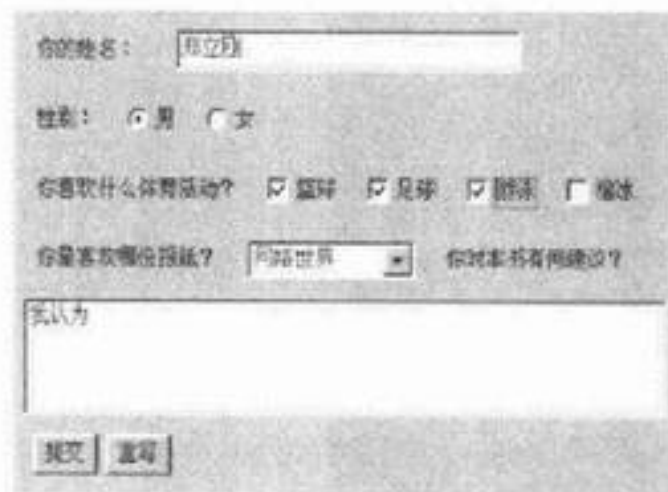


图 7.17 综合使用布局

在这个程序中,加入了若干个 Panel 类,在屏幕上显示在同一行中的组件都属于同一个 Panel。程序中默认的布局是 FlowLayout,排列方式默认是居中,程序中改成左对齐。图 7.17 是执行后的结果。程序在各个 Panel 中使用相同的布

局,当然也可将各个 Panel 设置成不同的布局,这样效果会有所不同。通过综合使用布局,可以设计出各种各样的版面。这个程序只是一个综合使用布局的简单示例,相信您会在实践中会有更多的创新。

7.3 多线程小程序

本节我们介绍 Java 的一个重要特性——多线程。我们首先介绍多线程的概念,其次讨论线程的创建、方法、同步等问题,最后通过一个在网页上显示动画的例子来说明多线程在小程序上的应用。学习完本节后,相信您就会明白许多精彩实例是怎样做出来的,因为那些精彩实例都涉及到多线程的应用。

7.3.1 什么是多线程

多线程的含义是可将程序任务分成几个可以同时进行的子任务。在网络编程中,您会发现很多功能是可以并发执行的。多线程是与单线程比较而言的。普通的 Windows 程序采用单线程程序结构,其工作原理是:主程序有一个消息循环,不断从消息队列中读入消息来决定下一步所要干的事情,一般是一个子函数,只有等这个子函数执行完返回后,主程序才能接收另外的消息来执行。例如,子函数功能是读一个网络数据或文件,那么只有等读完这个数据或文件才能接收下一个消息。在执行这个子函数过程中其他的什么也不能干。往往读网络数据和等待用户输入有很多时间处于等待状态。多线程将任务分成多个并发任务后,就可以解决这个问题。

我们可以通过计算机在同一时刻做多件事情来理解多线程的概念。举例来说,您可以坐在计算机旁一边通过 IE 浏览器从某个 Web 站点下载资料,一边用 Word 字处理程序写信,一边用 CD 播放器来听音乐。这是三个不同的应用程序,但它们共享同一个 CPU,这就很像多线程。不过,真正的多线程是在一个应用程序中共享 CPU。真正的多线程是这样的,在使用 Word 字处理程序打印一份文件的同时,还可以同时使用该应用程序写另一份文件。因为 Word 字处理程序是多线程的,所以能够利用 Word 字处理程序同时做几件事。具有多线程的应用程序叫做多线程应用程序。

我们下面将介绍 Java 语言同样具备多线程的特性,这一特性能够使我们用 Java 语言来开发多线程应用程序和小程序。

7.3.2 在小程序中创建线程

Java 虚拟机允许一个程序有多个线程并发执行,其中每个线程有一个优先级,优先级高的线程比优先级低的线程先执行。每个线程都可以被标注为后台监视器(daemon),即该线程暂时不执行,但一直监视系统有无发出让自己重新启动的消息,一旦系统发出重新启动线程的消息,这个线程就立即开始执行。如果没有重新启动的消息,这个线程就一直等待。

一般情况下,Java 应用程序和小程序都是虚拟机自动创建线程。如果想要应用程序和小程序独立做事情的话,您必须自己创建线程。创建线程有两种方法,一种方法是扩展 `java.lang.Thread` 类;另一种方法是使用 `java.lang.Runnable` 接口。从名字上就可以看出,Thread 类和 Runnable 接口都是在 `java.lang` 包中定义的,在 Visual J++ 的对象浏览器(Object Browser)中可以查看它们的详细信息。

扩展 `java.lang.Thread` 类

这个方法是声明一个 Thread 类的子类,这个子类将重载 Thread 类的 `run()` 方法。这样,这个子类的对象就成为一个线程对象。下面,我们用扩展 `java.lang.Thread` 类的方法创建一个线程。



```
class Simple
{
    int Numcount = 0;
    class MyThread extends Thread
    {
        public void run()
        {
            Numcount = Numcount + 1;
        }
    }
    public Simple()
    {
        Thread t = new MyThread();
    }
}

Simple s;
```

这个例子中我们首先声明了一个 Simple 类,在 Simple 类中又通过扩展 Thread 类(`extends Thread`),声明了一个 Thread 类的子类 MyThread,之后定义了一个 Simple 类的对象 s。定义对象 s 时,s 的成员变量 Numcount 被赋值为 0,同时

s 的构造函数被自动调用,于是执行“Thread t = new MyThread();”语句,这就创建了一个 MyThread 类的对象 t。对象 t 就是一个线程,它具有 run() 方法,但这个方法没有被调用,因此线程 t 只是被创建,而没做什么事情。

使用 java.lang.Runnable 接口

使用 Runnable 接口很简单,这种方法是通过声明实现 Runnable 接口的一个类来创建线程的,这样,这个类就能使用 run() 方法。Runnable 接口只有一个方法 run(),Runnable 接口的定义如下:

```
public interface Runnable
{
    public void run();
}
```

Thread 类有一个以 Runnable 对象为参数的构造方法。任何类都能实现 Runnable 接口。调用具有 Runnable 对象的 Thread 构造方法就可以创建一个 Thread 对象。下面使用 Runnable 接口来改写前面例子代码:

```
class Simple2 implements Runnable
{
    int Numcount = 0;
    public void run()
    {
        Numcount = Numcount + 1;
    }
    public Simple2()
    {
        Thread t = new Thread(this);
    }
}
Simple2 s;
```

这个例子中我们首先声明了一个 Simple2 类,在 Simple 类的声明中实现 Runnable 接口,之后定义了一个 Simple2 类的对象 s。定义对象 s 时,s 的成员变量 Numcount 被赋值为 0,同时 s 的构造函数被自动调用,于是执行“Thread t = new Thread(this);”语句,这就创建了一个线程 t。由于类 Simple2 使用了 Runnable 接口,它就可以重载 run() 方法。这样,Thread 对象的构造方法获得 this 引用,Thread 对象 t 就可以从 this 引用调用 run() 方法。

在具体应用中,采用哪种方法来创建线程要视情况而定。通常,当一个线程已继承了另一个类时,就应该用第二种方法来构造,即实现 Runnable 接口。例如,由于 Java 小程序继承了 Applet 类,所以如果在 Java 小程序中创建线程,就应

该使用 `java.lang.Runnable` 接口。

7.3.3 线程的方法

`Thread` 类中有 7 个重要的方法：`start()`、`run()`、`suspend()`、`resume()`、`sleep()`、`interrupt()` 和 `stop()` 方法，分别表示线程的启动、执行、暂停、重新运行、睡眠、唤醒和停止。

`start()` 和 `run()` 方法

`start()` 和 `run()` 是最主要的 `Thread` 方法，它们的定义为：

```
public void start();
public void run();
```

调用 `start` 方法可以启动线程，在前面的例子 `Simple` 类的构造函数中加一句“`t.start()`”，这样定义对象 `s` 时，线程 `t` 就会被创建并且启动。



```
class Simple
{
    int Numcount = 0;
    class MyThread extends Thread
    {
        public void run()
        {
            Numcount = Numcount + 1;
        }
    }
    public Simple()
    {
        Thread t = new MyThread();
        t.start();
    }
}
Simple s;
```

调用 `start()` 方法后，`Thread` 对象 `t` 获得 Java 虚拟机创建的控制流程，该控制流程执行 `run` 方法中的代码。一旦 `run` 方法执行完毕，Java 虚拟机就把线程清除，并标识它，以便进行自动垃圾收集（Java 的一种与内存管理有关的机制）。

`suspend()` 和 `resume()` 方法

`suspend()` 方法暂停 `Thread` 对象的控制流程，而 `resume()` 方法让控制流程重

新运行起来,它们的定义为:

```
public void suspend();
public void resume();
```

`suspend()`方法可以被自己和别的线程调用,通常被如鼠标单击事件处理程序等线程调用。例如由线程实现动画显示的暂停,就可以通过鼠标单击事件(对应一个线程)调用 `suspend()` 方法。线程暂停后,该线程就不能做任何事情,甚至没有办法调用 `resume()` 方法使自己重新启动,直到有别的线程调用了 `resume()` 方法(比如可能是另一个单击事件处理程序)。

`sleep()`方法

`sleep()`方法使控制流程暂停给定的一段时间,以毫秒为单位。它的定义为:

```
public static void sleep(long millis) throws InterruptedException;
```

`sleep()`方法有些特殊的语法。

首先,`sleep()`方法是静态的,不需要特定的 `Thread` 对象就可以调用它。即使应用程序中没有引用 `Thread` 对象,线程调用 `Thread.sleep()` 后也转入睡眠状态。在任意方法中,都可以调用 `sleep()` 方法。

其次,`sleep()`方法有一个和它关联的异常 `InterruptedException`,关键字 `throws` 标明这种关联。关于异常,请参看有关“异常处理”的内容。

`interrupt()`和 `stop()`方法

`interrupt()`方法唤醒正在睡眠的线程,`stop()`方法使线程停止,它们的定义为:

```
public void interrupt();
public void stop();
```

如果某个线程调用了 `sleep()` 方法而睡眠,可以调用 `interrupt()` 方法来唤醒它,`interrupt()` 方法用 `InterruptedException` 强制使 `sleep()` 的调用提前返回。



注意 `stop()`方法并没有消灭线程,只是停止了线程的执行,并且这个线程不能用 `start()` 方法重新启动。很多复杂的线程需要我们控制每个线程,在这种情况下会使用到 `stop()` 方法。一个线程已经启动而且没有停止,则被认为是激活的。如果需要,可以用在 `Thread` 类的程序接口中提供的 `isAlive()` 方法测试线程是否被激活。如果线程已被启动并且未被终止,那么 `isAlive()` 返回 `true`。如果返回 `false`,则该线程是新创建或是已被终止的;若返回 `true`,则该线程是可运行或是不可运行的,但是不能作进一步的分辨。

7.3.4 同步

多线程提供了程序的异步执行的功能,也就是说每个线程都包含了运行时所需要的数据或方法,而不需要外部的资源或方法,也不必关心其他线程的状态或行为。但是经常有一些同时运行的线程需要共享数据,共享是多线程程序中最重要内容。这时就需要实现同步来得到预期结果,在必要时必须提供一种同步机制。

例如,用 Word 字处理应用程序已经写好了一封信,并用后台线程打印。至于另一封信,您决定不另外写,而只是在前一封信的基础上做一些修改。由于正在前台的编辑窗口修改第一封信,所以不希望后台线程使用它,即前后台的这两个线程要独立地各做各的事,避免冲突。字处理应用程序是这样解决冲突的,它创建前台编辑窗口中的内容副本给后台打印,前台以后做的修改不影响副本。这就是同步处理。

为达到同步的目的,大多数多线程系统用一种叫监视器(monitor)的机制实现了进程间的同步执行。可以将监视器看作是一个很小的盒子,它只能容纳一个线程。当一个线程进入一个监视器时,所有其他线程必须等到这个线程退出监视器后才能进入。监视器可以保护共享的数据不被多个线程同时操作。

大多数多线程系统将监视器设计成对象,Java 没有定义 Monitor 对象,而是提供了一种更清晰的解决方案。在 Java 中,如果有多个不同的线程要更新、使用同一些数据时,可以用关键字 `synchronized` 来避免冲突。关键字 `synchronized` 是方法的修饰符。当调用了有 `synchronized` 修饰的方法后,Java 虚拟机阻塞该对象上任何别的同样有 `synchronized` 修饰方法的调用,直到这一个方法调用完成。这样,每个对象通过将它们的成员函数定义成 `synchronized` 来定义自己的显式监视器。于是,当一个线程执行一个对象的 `synchronized` 函数时,其他任何线程都不能调用这个对象的 `synchronized` 函数。

我们把系统中使用某类资源的线程称为消费者,产生或释放同类资源的线程称为生产者,下面我们来讨论关于线程的同步问题的一般模型,即生产者-消费者问题。我们可以想象在一个 Java 的应用程序中,生产者线程向文件中写数据,消费者从文件中读数据。这样,在这个程序中同时运行的两个线程共享同一个文件资源。通过这个例子我们来了解如何使它们同步。

在这个例子中,生产者产生从 0 到 9 的整数,将它们存储在名为 `CubbyHole` 的对象中并打印出这些数。然后调用 `sleep()` 方法使生产者线程在一个随机产生的 0 到 100 秒的时间段内睡眠。



```
class Producer extends Thread {
    private CubbyHole cubbyhole;
    private int number;
    public Producer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            cubbyhole.put(i);
            System.out.println("Producer #" + this.number + " put: " +
                i);
            try {sleep((int)(Math.random() * 100));}
            catch (InterruptedException e) {}
        }
    }
}
```

消费者线程则不断地从 CubbyHole 对象中取这些整数:



```
class Consumer extends Thread {
    private CubbyHole cubbyhole;
    private int number;
    public Consumer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }

    public void run() {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = cubbyhole.get();
            System.out.println("Consumer #" + this.number + " got: " +
                value);
        }
    }
}
```

在这个例子中,生产者与消费者通过 CubbyHole 对象来共享数据。但是不论是生产者线程还是消费者,线程都无法保证生产者每产生一个数据,消费者就能及时取得该数据并且只取一次。通过 CubbyHole 中的 put() 和 get() 方法只能保证较低层次的同步,让我们来看一看可能发生的情况。

第一种情况是如果生产者比消费者快,那么在消费者来不及取前一个数据之前,生产者又产生了新的数据,于是,消费者很可能就会跳过前一个数据,这样

就会有下面的结果:



```
...
Consumer #1 got: 3
Producer #1 put: 4
Producer #1 put: 5    // Producer 连续产生两个数据
Consumer #1 got: 5    // Consumer 这时应该为 got: 4
...
```

第二种情况则反之,当消费者比生产者快时,消费者可能两次取同一个数据,可能会产生下面的输出结果:



```
...
Producer #1 put: 4
Consumer #1 got: 4    //Consumer 连续取得两个数据
Consumer #1 got: 4    // Consumer 这时不应该为 got: 4
Producer #1 put: 5
...
```

上面两种输出结果都不是我们所希望的,生产者写一个数,消费者就取这个数。像这种异步执行的多线程,由于希望同时进入同一对象中而发生错误结果的情况称为竞争条件。

为了避免上述情况发生,就必须使生产者线程向 CubbyHole 中存储数据和消费者线程从 CubbyHole 中读取数据同步起来。为达到这一目的,程序中采用了监视器、notify()方法和 wait()方法。下面我们来介绍有关内容。

监视器 (monitor)

像 CubbyHole 这样被多个需同步的线程共享的对象我们称它为条件变量。这里的条件变量就相当于一个监视器,Java 语言正是通过使用监视器来实现同步。

生产者-消费者问题中的类 CubbyHole 中提供了两个同步的方法,put()方法用来改变 CubbyHole 中的数据,get()方法则用来取数据。这样系统就把每个 CubbyHole 类的实例与一个 monitor 相对应。



```
class CubbyHole {
    private int seq;
    private boolean available = false;

    public synchronized int get() {
        while (available == false) {
            try {
```



```

        wait(); // waits for notify() call from Producer
    }
    catch (InterruptedException e) {}
}
available = false;
notify();
return seq;
}

public synchronized void put(int value) {
    while (available == true) {
        try {
            wait(); // waits for notify() call from consumer
        }
        catch (InterruptedException e) {}
    }
    seq = value;
    available = true;
    notify();
}
}

```

CubbyHole 类有两个变量, seq 是 CubbyHole 中当前内容, 布尔变量 available 指示当前内容是否可以取出。只有当 available 是真时, 消费者才能取数据。为实现两线程同步, 必须保证:

- ◆ 消费者接收数据的前提是 available 为真, 即数据单元内容不空。
- ◆ 生产者发送数据的前提是数据单元内容为空。

在这个例子中, 通过调用对象的 notify() 和 wait() 方法来保证 CubbyHole 中的每个数据只被读取一次。

notify() 方法

在 get() 方法返回前调用 notify(), 它用来选择并唤醒等候进入监视器的线程。如果消费者线程调用了 get() 方法, 那么在整个执行过程中它将占据 monitor, 在 get() 方法结束前调用 notify() 方法来唤醒处于等待状态的生产者线程。这样, 生产者线程就占据了 monitor 并继续执行。

put() 方法与 get() 相似, 在一个线程结束前唤醒另一个。与 notify() 方法类似的还有 notifyAll() 方法, 不同的是它唤醒所有等待的线程, 这些线程中的一个经过竞争进入 monitor, 其他的继续等待。

wait()方法

wait()方法使当前线程处于等待状态,直到别的线程调用 notify()方法来通知它。

get()方法包含了一个 while 循环,结束条件是 available 为真。如果 available 为假的话,消费者就知道生产者还没有产生新的数据,将继续等待。while()循环中调用 wait()方法,等待生产者线程发送消息。当 put()方法调用 notify()时,消费者线程被唤醒并继续 while()循环。put()方法中的 wait()方法作用相同。

可能出现的问题是,在 get()方法的开始,available 为假,消费者线程必须等待生产者发送数据,那么,如果消费者占据 monitor,生产者如何发送呢?同样,若数据未被消费者取走,而生产者占据 monitor,消费者怎样获得数据呢?在 Java 中是这样处理的,当一个线程进入等待状态后,monitor 就会自动释放,而当它被唤醒后,该线程又占据 monitor。这样就使等待的线程有机会进入 monitor。

下面我们来看一看主程序及输出结果:



```
class ProducerConsumerTest {
    public static void main(String args[]) {
        CubbyHole c = new CubbyHole();
        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);
        p1.start();
        c1.start();
    }
}
```

输出结果如下:



```
Producer #1 put: 0
Consumer #1 got: 0
Producer #1 put: 1
Consumer #1 got: 1
Producer #1 put: 2
Consumer #1 got: 2
Producer #1 put: 3
Consumer #1 got: 3
Producer #1 put: 4
Consumer #1 got: 4
Producer #1 put: 5
Consumer #1 got: 5
```

由输出结果,我们可以看到生产者和消费者两个线程实现了同步。需要注意的是由于系统资源有限,程序中多个线程互相等待对方资源,而在得到对方资

源前不会释放自己的资源,造成都想得到资源而又都得不到,线程不能继续发展。这就是死锁问题,这里我们不做进一步的探讨。

7.3.5 多线程在小程序中应用的例子

我们下面要做一个 Java 小程序使用多线程的例子,在网页上显示一个简单的动画,图片将从上面滑下,反复进行。在 Web 浏览器中显示的结果如图 7.18 所示。



图 7.18 使用多线程显示动画

下面是源代码:



```
import java.awt.*;
import java.applet.*;

public class Logo extends Applet implements Runnable
{
    Image img;
    Thread thd = null;
    int i;
    int imgWidth = 120;
    int imgHeight = 277;
    public void run()
    {
        img = getImage(getCodeBase(), "test.jpg");
        if (img != null)
```

```

        }
        i = imgHeight;
        repaint();
        while (true)
        {
            try {Thread.sleep(1000);} catch (InterruptedException
            e){}
            i = 0;
            while (i < imgHeight)
            {
                repaint();
                try {Thread.sleep(50);} catch (InterruptedException e){}
                i += 4;
            }
        }
    }

    public void update(Graphics g)
    {
        if (img != null)
        {
            g.clipRect(0, 0, imgWidth, i);
            g.drawImage(img, 0, i - imgHeight, null);
        }
    }

    public void start()
    {
        if (thd == null)
        {
            thd = new Thread(this);
            thd.start();
        }
    }

    public void stop()
    {
        thd = null;
    }
}

```

run()方法表示了一个新的线程。在该单独的线程中,代码装入一幅图像,通过在循环过程中每隔 50 毫秒在新位置重画它来实现动画。然后代码等待一秒钟,重复动画。

每次代码在新位置处重画位图,它都要调用 repaint。该函数调用可重载的

applet 的父类 `java.awt.Component` 的 `update(java.awt.Graphics)` 方法。`update` 方法与 `paint` 方法是相同的,但 `paint` 方法在绘图前要清除窗口,而 `update` 方法不清除(把 `update` 方法改名为 `paint`,观察有什么不同)。此处,`update` 方法只简单地在由 `i` 指定的位置处绘制图像。

如果查看运行效果,可参见随书光盘的 `examples\chap07\ex07thread` 下的 `Page1.htm`。还可以尝试在页面上加上按钮来实现动画的开始、暂停或者改变动画移动的方向。

如果要制作一组 10 张图片组成的动画,会发现所要做的只是把这 10 张图片装到一个图像数组中,然后用 `update` 方法按顺序绘制它们。



更上一层楼

本章中我们对 Java 小程序编程做了进一步的介绍,学习了运用小程序的界面组件进行动态网页设计、对 Web 页面上的小程序界面组件进行布局管理、创建多线程小程序等内容。有关 Java 小程序编程的具体应用还有很多方面,例如小程序的其他动画技术、小程序的上下文、小程序之间的通信、给小程序增加音频等,读者可以根据自己的兴趣去查看相应的资料,编写相应的代码,使自己的小程序更具个性化。

第 8 章 Java 应用程序编程入门

知识要点：

- ◆ 创建和显示启动屏幕
- ◆ 制作屏幕保护程序
- ◆ 练习菜单操作：创建菜单栏和环境菜单
- ◆ 利用 ToolBar 控件创建工具栏
- ◆ 利用 StatusBar 控件创建状态栏

在本章中，我们将学习 Java 应用程序入门级的知识——窗体和窗体附件，以及如何创建和使用窗体和窗体附件。我们将动手创建启动屏幕和屏幕保护程序两种特殊的窗体。讨论创建菜单、工具栏、状态栏及其事件处理程序。



光盘 参阅本书配套光盘【建立文本编辑器】部分，可交互学习与本讲相关的知识。

8.1 创建和显示窗体

本节我们将使用窗体设计器(Forms Designer)来学习创建和操纵应用程序窗口。在完成本节之后,您将能够创建窗体,并能够处理各种与窗体相关的问题。

8.1.1 窗体简介

在 Visual J++ 中,窗体(Form)是窗口或对话框的可视化表现,窗体上可以放置按钮、菜单、工具栏、状态栏、复选框等可视化控件,也可以放置数据库控件、计时器控件等非可视化控件。在 80 年代,人们还在使用 DOS 操作系统,那时程序开发人员使用 Basic、Turbo C、Pascal 等语言编写应用程序。为了显示一个简单的窗口,程序开发人员不得不自己编写大量代码来处理窗口的外观、消息、事件等等,因此程序员的工作中相当大的一部分是近乎于体力劳动。随着 Windows 操作系统的出现,软件开发工具开始向图形化方向发展,这种情况逐渐有了改观。这些软件开发工具使开发窗口与使用控件的工作简化为只需将它们拖放到可视的设计器中。创建窗口和添加控件已经不再费力,人们可以把主要精力放在程序本身的逻辑设计上,这就是所谓的“可视化编程”。

Visual J++ 可以使程序员使用 Java 语言编写 Windows 应用程序,Visual J++ 的集成开发环境(IDE)和“Windows 基类”(WFC)使 Java 程序员能够进行快速、方便的 Windows 应用程序设计。Toolbox 中的 WFC Controls 工具箱提供了用于创建强大应用程序的一系列控件,窗体设计器(Forms Designer)可以用来快速地设计应用程序的用户界面。无论您是设计应用程序的窗体,还是在窗体上放置和编辑 WFC 控件,或者仅仅是将窗体的事件处理程序同 WFC 控件关联起来,窗体设计器都会为您节省大量的时间和精力。

下面的实例演示了创建和操作窗体的方法,并且提供了在设计应用程序时使用窗体所需的一些知识。

如今,没有用户界面的应用程序是比较少见的,所以 Windows 程序开发者必须掌握如何创建和操作窗体。Visual J++ 集成开发环境提供了 3 个与创建窗体密切相关的工具:【窗体设计器】(Forms Designer)、【工具箱】(Toolbox)和【属性窗口】(Properties)。Forms Designer 是用来设计用户界面的窗体,Toolbox 提供可以拖放到窗体上的控件,Properties 窗口用来设置窗体或放置在窗体上的控件的属性及事件处理程序。如果您熟练掌握了这 3 个工具,将发现设计 Windows 应用程序是相当简单的。

8.1.2 创建和显示窗体的实例

为了帮助您理解创建和显示窗体的方法,下面将介绍制作启动屏幕的窗体实例。所谓启动屏幕,就是一个软件或程序在启动开始时显示的一个窗体,目的是在出现主窗口之前,显示有关本软件或程序的一些信息(软件名称、版本号、版权声明、授权用户、警告信息等)。启动屏幕一般只出现几秒钟就自动消失。

Visual J++ 在启动的时候就有一个启动屏幕,这里我们做一个简单的启动屏幕,图 8.1 显示了该启动屏幕的外观。



图 8.1 启动屏幕

主窗口很简单,只显示一行字(如图 8.2 所示),因为我们的目的是制作启动屏幕。

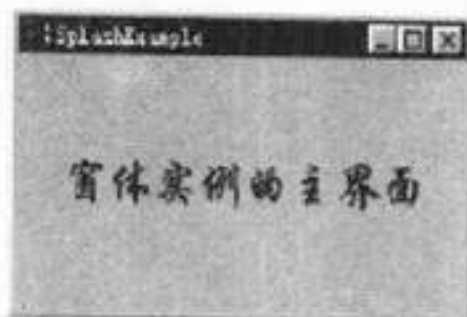


图 8.2 窗体实例的主窗口

通过制作启动屏幕的过程,可以很好地了解创建和显示窗体的方法。在该

实例中,将创建一个能够提供可以定义启动屏幕外观的方法的启动屏幕。完成以下的步骤后,将得到一个启动屏幕窗体,可以在自己的应用程序中进行修改。

8.1.3 创建工程 and 应用程序的主窗体

首先需要在 Visual J++ 中为启动屏幕创建一个工程。该工程只需要两个窗体,所以我们将使用 Visual J++ 提供的标准 Windows 应用程序模板。如果是首次在 Visual J++ 创建该工程,那么在创建工程时首先要指定工程的路径,然后输入“SplashExample”作为该工程的名称。窗体的标题默认为“Form1”。下面来更改窗体默认的标题。

在 Project Explorer 中,右击窗体名称,然后从快捷菜单中选择 Rename 命令,将文件名改成 SplashExample.java,按 Enter 键,保存所做的更改。

在代码编辑器中打开该窗体,用 SplashExample 替换所有的 Form1。您可以用 Edit 菜单中的 Find And Replace 命令来简化这项工作。然后,从 WFC Control 工具箱中拖动一个标签控件(Label)到该窗体上,更改 Label 的 text 属性为“窗体实例的主界面”,更改 Label 的 font 属性为您所喜欢的字体。现在,窗体文件就准备好了。该窗体将是应用程序的基础。

为了在代码编辑器和窗体设计器间切换,可以使用 View 菜单中的 Code 命令和 Designer 命令,其中 Code 命令是用于切换到代码编辑器查看代码,Designer 命令是用于切换到窗体设计器设计窗体。也可以直接按功能键 F7 和 Shift + F7 来实现相同的功能。



提示 实现代码编辑器和窗体设计器间切换最方便的方法是使用右键。可以在 Project Explorer 中,右击窗体名称,然后从快捷菜单中选择 View Code 或 View Designer 命令来实现二者的切换(如图 8.3 所示)。

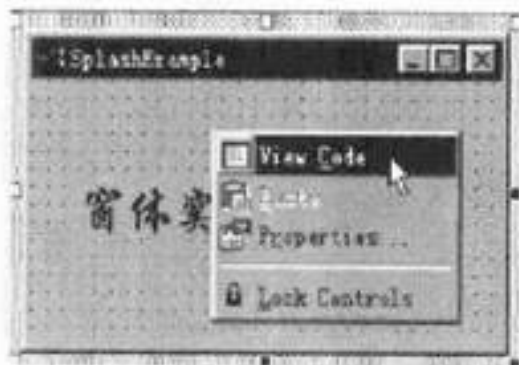


图 8.3 右键实现窗体设计器到代码编辑器的切换

8.1.4 创建启动屏幕窗体

创建完工程 and 应用程序的主窗体后,就可以开始创建启动屏幕窗体了。在工程中添加一个新窗体,将它命名为“SplashScreen.java”。这个窗体将成为启动屏幕,而且实例的绝大部分代码会用于此。该窗体显示在 Forms Designer 中后,需要按照表 8.1 调节其大小。启动屏幕只需使用两个 WFC 控件: PictureBox 和 Timer。将这两个控件拖放到窗体上,然后按照表 8.1 的数据设置它们的属性。

可以通过选择 Project 菜单的 Add Item 命令,将窗体或任何其他项添加到工程中。也可以通过在 Project Explorer 中右击工程来添加它们。从快捷菜单中选择 Add 命令,然后选择窗体或其他项。

表 8.1 SplashScreen.java 中的组件及其属性设置

组件类型	属性	设置值
Form	size.x	455
	size.y	310
	controlBox	False
	borderStyle	FixedDialog
	minimizeBox	False
	maximizeBox	False
	text	" * !"
	showInTaskbar	False
	startPosition	CenterScreen
	startPosition	CenterScreen
PictureBox	name	PicLogo
	size.x	145
	size.y	145
	sizeMode	StretchImage
Timer	name	TmrSplash
	interval	2500

窗体的 controlBox、borderStyle 及 text 属性的值确定了启动屏幕为固定边框、无标题栏的窗口。showInTaskbar 属性被设置为 false,以避免启动屏幕被当作任务显示在 Windows 的任务栏内。尽管启动屏幕窗体的该项属性被如此设置,但这并不妨碍应用程序的主窗体被显示在任务栏内。picLogo 的 sizeMode 属性允许拉伸徽标图片以适合标准尺寸。这就使得我们可以通过拉伸图片而调整图片框在窗体上的位置和大小。tmrSplash 的 interval 属性的初始值被设置为 2500 毫秒(2.5 秒),允许启动屏幕的使用者将此间隔设置成任何需要的有效值。图 8.4 显示了该窗体在设计时的样子。

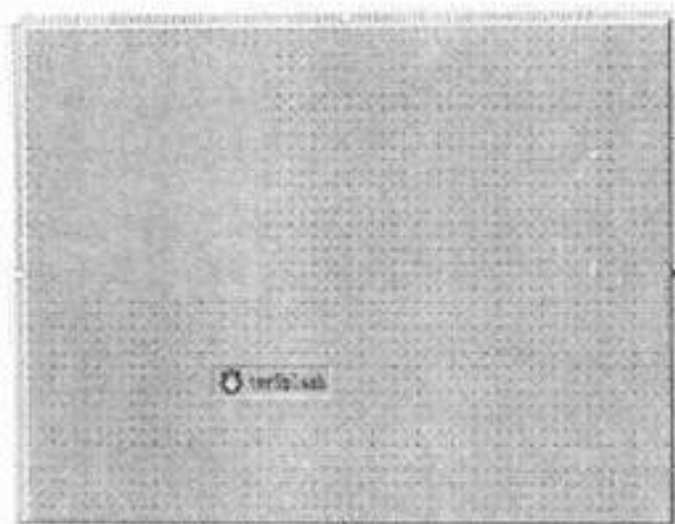


图 8.4 设计时的 SplashScreen.java

需要添加到窗体中的最后一项内容是 WFC 的 `com.ms.wfc.io` 包的一个导入语句。我们将使用该包中的一些静态方法来确定图片文件在系统中的位置。您应当将下面的代码放置在 `SplashScreen.java` 源文件的所有其他导入语句后：

```
import com.ms.wfc.io.*;
```

8.1.5 将代码添加到启动屏幕的窗体中

现在启动屏幕实例的两个窗体都已经创建,并放到了编辑器中。我们马上就来添加一些代码。这个启动屏幕实例的目的不仅是向您展示如何创建并显示一个窗体,而且还要说明如何使其所有内容都封装在一个类中。正像在所有其他面向对象的程序设计语言中的情况一样,在 Visual J++ 中您的目标应当是编写自给自足的类,使它能够轻易应用于其他应用程序。为了使 `SplashScreen` 类重用性更高,需要利用一组公用的方法来访问和设置要在启动屏幕上显示的文本信息和图形。

添加到启动屏幕窗体的第一段代码是一组用来存储将要在启动屏幕上显示的信息的私有成员(`private`)变量的声明。我们还要添加一组设置这些私有成员变量的值的赋值函数,如创建的启动屏幕显示产品名称、版本号、版权声明、警告信息,以及授权用户等应用程序信息。同时,用户可以设置背景和徽标图像。在 `SplashScreen` 类中添加以下代码:



//声明私有成员变量

```
private String m_sProduct = new String(); //产品名称
private String m_sVersion = new String(); //版本号
private String m_sCopyright = new String(); //版权声明
private String m_sWarningText =           //警告信息
    new String("本软件产品受著作权法和国际公约的保护!");
private String m_sUserName = new String(); //授权用户名
private String m_sCompanyName = new String(); //授权公司名
private String m_sBackPath = new String(); //背景图片路径
private Bitmap m_bmBackSplash;           //背景图片
```

//设置启动屏幕持续显示的时间

```
private void setSplashDelay(int delay)
{
    if (delay > 0)
        tmrSplash.setInterval(delay);
}
```

//设置启动屏幕上的产品名

```
public void setProduct(String productName)
{
    m_sProduct = productName;
}
```

//设置启动屏幕上的版本号

```
public void setVersion (String versionText)
{
    m_sVersion = versionText;
}
```

//设置启动屏幕上的版权声明

```
public void setCopyright(String copyRight)
{
    m_sCopyright = copyRight;
}
```

//设置启动屏幕上的授权用户名

```
public void setUserName(String userName)
{
    m_sUserName = userName;
}
```

//设置启动屏幕上的授权公司名

```
public void setCompany(String company)
{
}
```

```
m_sCompanyName = company;
|

//设置启动屏幕上的警告信息
public void setWarningText(String warning)
|
    //判断警告信息是否为空
    if (warning.length() > 0)
    |
        m_sWarningText = warning;
    |
    //如果参数为"NA",那么设置警告信息为空
    if (warning.equals("NA") == true)
    |
        m_sWarningText = "";
    |
|

//设置载入图片的路径
public void setLogoImage(String logoImage)
    throws IOException
|
    //判断图片路径非空,并且图片文件存在
    if (logoImage.length() > 0 && File.exists(logoImage) == true)
    |
        try
        |
            //载入图片到 picturebox
            picLogo.setImage(new Bitmap(logoImage));
            //显示图片
            picLogo.setVisible(true);
        |
        catch (IOException ex)
        |
            MessageBox.show("载入图片时发生错误!"
                            + "请检查图片的路径和文件名",
                            "Splash Example",
                            MessageBox.ICONERROR
                            + MessageBox.OK);

            //设置图片为不可见
            picLogo.setVisible(false);
            return;
        |
    |
|
```



```

//设置背景图片路径
public void setSplashBackground(String backImage)
{
    m_sBackPath = backImage;
}

public SplashScreen(String product,
                    String version,
                    String copyright,
                    String userName,
                    String company,
                    String warning)
{
    this();
    setProduct (product);
    setVersion(version);
    setCopyright(copyright);
    setUserName(userName);
    setCompany (company);
    setWarningText(warning);
}

```

大部分代码非常简单。私有成员变量存储了启动屏幕要显示的字符串和图片的信息。除了 `m_sWarningText` 以外,所有字符串都默认为空值。添加默认字符串应该非常简单。如果您不想显示任何警告信息,那么只需用 `NA` 值作为参数调用 `setWarningText` 方法即可。要保留默认警告信息,可以用空字符串作为参数调用 `setWarningText` 方法。有一个私有 `String` 成员变量并不在启动屏幕上显示信息,而是存放启动屏幕背景图片的路径(当用户需要指定要显示的背景图片时有用)。图片变量此时未被初始化,它保留到等用户指定了用于显示的图片时再完成。

大部分赋值方法都很简单:接受一个 `String` 参数,然后将它赋给相应的私有成员变量。赋值方法的两个例外是 `setLogoImage` 和 `setSplashDelay` 方法。`setLogoImage` 方法接受用于显示前面添加到窗体中的 `picLogo PictureBox` 控件中图片的文件路径。我们在此方法的声明中添加了 `throws` 语句,以进行 `IOException` 异常处理。`throws` 语句对于执行捕获 I/O 异常的异常处理代码是必需的。I/O 异常可能发生在向 `picLogo PictureBox` 控件加载图片的时候。该方法的代码首先判断参数传入的路径是不是空字符串。如果存在路径名,那么代码用 `File` 类的 `exists` 方法判断是否有相应的文件。如果文件存在,那么该方法将返回一个为 `true` 的 `Boolean` 值。如果两个条件都满足(即路径和文件都存在),那么代码就将图片文件加载到 `Bitmap` 对象中,该对象被用于 `picLogo PictureBox` 控件。如果因为一些

原因 picLogo 没有被正常加载,那么代码中的捕获部分将被执行,并显示错误信息,picLogo 将保持不可见。因为显示徽标并不是特别重要的一件事情,所以此处的异常处理代码相当简单。

setSplashDelay 方法是私有的赋值方法,它用来设置启动屏幕显示的持续时间。该方法被声明为私有是因为它是 showSplash 方法的辅助方法(下一步将会讨论)。我们还给 SplashScreen 类添加了一个构造函数,以便尽可能简单地将文本信息加载到启动屏幕中。新构造函数在第一行调用默认构造函数以执行其代码。虽然在本实例中这样做并不会显著减少代码量,但还是建议您应该在构造函数中调用被重新加载的版本(或在其他类似的情形下)。以便能够使用在父类构造函数中定义的代码,当程序很大,构造函数也很大的时候,这样做就很有用了。

8.1.6 添加启动屏幕窗体的方法和事件处理程序

在使用代码设置启动屏幕信息后,添加一些方法和事件处理程序,以使这些信息显示在启动屏幕上。我们还要添加设置启动屏幕持续时间和显示窗体的代码。在 SplashScreen 类中添加以下代码:



```
//显示启动屏幕,参数 interval 决定了启动屏幕持续的时间
public void showSplash(int interval) throws IOException
{
    //设置启动屏幕持续的时间(毫秒)
    setSplashDelay(interval);
    //检查背景图片是否存在
    if (m_sBackPath.length() > 0
        && File.exists(m_sBackPath) == true)
    {
        try
        {
            m_bmBackSplash = new Bitmap(m_sBackPath);
        }
        catch (IOException ex)
        {
            MessageBox.show("载入图片时发生错误!"
                + "请检查图片的路径和文件名",
                "Splash Example",
                MessageBox.ICONERROR
                + MessageBox.OK);
        }
    }
}
```

```
|
|
//启用 timer 控件
tmrSplash.setEnabled(true);
this.showDialog();
|

//关闭启动屏幕
public void closeSplash(Object sender, Event e)
|
    tmrSplash.setEnabled(false);
    this.dispose();
|

//绘制启动屏幕
public void paintSplash(Object sender, PaintEvent e)
|
    Graphics gdi;
    if (m_bmBackSplash == null)
    |
        gdi = this.createGraphics();
    |
    else
    |
        gdi = m_bmBackSplash.getGraphics();
    |

    gdi.setOpaque(false);
    gdi.setFont(new Font("Arial", 12.0f,
                           FontSize.CHARACTERHEIGHT,
                           FontWeight.BOLD,
                           false, false, false));

    //绘制授权信息
    gdi.drawString("本窗体实例允许下列公司或个人使用:", 170, 40);

    //绘制用户名
    if (m_sUserName.length() > 0)
        gdi.drawString(m_sUserName, 170, 60);
    //绘制公司名
```

```

        if (m_sCompanyName.length() > 0)
            gdi.drawString(m_sCompanyName, 170, 90);

        //设置版本号的字体
        gdi.setFont(new Font("Arial", 18.0f));
        //绘制版本号
        if (m_sVersion.length() > 0)
            gdi.drawString(m_sVersion, 170, 210);

        //设置版权信息的字体
        gdi.setFont(new Font("Arial", 14.0f,
                               FontSize.CHARACTERHEIGHT,
                               FontWeight.BOLD,
                               false, false, false));
        //绘制版权信息
        if (m_sCopyright.length() > 0)
            gdi.drawString(m_sCopyright, 170, 240);

        //设置警告信息的字体
        gdi.setFont(new Font("Arial", 12.0f,
                               FontSize.CHARACTERHEIGHT,
                               FontWeight.BOLD,
                               false, false, false));
        //绘制警告信息
        if (m_sWarningText.length() > 0)
            gdi.drawString(m_sWarningText, 10, 280);

        //设置产品名的字体
        gdi.setFont(new Font("Arial", 36.0f,
                               FontSize.CHARACTERHEIGHT,
                               FontWeight.NORMAL,
                               false, false, false));

        //先用灰色绘制产品名
        gdi.setTextColors(Color.GRAYTEXT);
        gdi.drawString(m_sProduct, 172, 172);
        //再用黑色绘制产品名,产生立体效果
        gdi.setTextColors(Color.BLACK);
        gdi.drawString(m_sProduct, 170, 170);
        //载入背景图片
        if (m_bmBackSplash != null) {
            gdi = this.createGraphics();
            gdi.drawImage(m_bmBackSplash, 0, 0);
        }
    }
}

```

在这段代码中,只定义了一个标准方法 `showSplash`。`showSplash` 用来显示启

动屏幕。该方法的唯一参数是一个用来设置启动屏幕显示时间的整数(毫秒)。showSplash 不仅负责显示窗体和设置窗体显示时间,而且还负责启动屏幕背景的 m_lmBackSplash 图片对象的初始化。图片一旦被显示,代码就启动计时器控件,并调用 showDialog 方法来显示启动屏幕。


在这一步中添加的另外两个方法都是事件处理程序。一般情况下,要创建事件处理程序,只需单击 Forms Designer 中的对象,其中包括了要在 Properties 窗口的 Event View 视图中处理和显示的事件。通过单击 Properties 窗口的  图标,可以进入该视图,如图 8.5 所示。



图 8.5 Properties 窗口的 Event View 视图

Properties 窗口的 Event View 视图显示了 Forms Designer 中当前选定对象的所有可能事件。要创建事件处理程序,可以双击要处理的事件名称,这样就会创建默认的事件处理程序名称。也可以通过在要处理的事件旁键入方法名来创建自己的处理程序名称。如果要使用类中已定义过的事件处理程序,那么只需单击事件旁边的下拉列表框,然后选择事件处理程序即可。对于一个给定的事件,列表中包含了该事件的所有处理程序的方法识别标识。如果要将上面的源代码复制到工程中,可以通过使用下拉列表框来将这些处理程序与其事件相对应。

不管采用哪种方式创建事件处理程序,当完成后,都将在窗体的类中看到这些事件处理程序,以及将处理程序同消息源连接起来的辅助代码。在本实例中,CloseSplash 和 PaintSplash 方法就是所要的处理程序。

CloseSplash 方法处理 tmrSplash 计时器控件的 timer 事件。这个事件处理程序负责在超出用户调入启动屏幕时定义的时间间隔后关闭启动屏幕,其代码使计时器控件失效并关闭窗口。

PaintSplash 方法处理窗体的 paint 事件,其中包含了本实例中较有意义的代码。paint 事件在每次需要重新绘制窗体时发生。对于多数窗体来说,当窗体被其他窗口遮挡,以及需要重新绘制时,paint 事件就会得以处理。由于本实例中启动屏幕只显示一次,所以 paint 事件只在窗体最初显示的时候发生。在 PaintSplash 事件处理程序中,要绘制启动屏幕的所有信息。绘制启动屏幕信息的工具是 Graphics 类。Graphics 类封装了 Windows“图形设备接口”(GDI)的所有功能。您可以用一个 Graphics 对象向该对象所属的窗体绘制图像、文本和图形。

在该方法的代码开始处,创建了一个名为 gdi 的 Graphics 实例。由于要留给启动屏幕的用户自己选择背景图像的机会,所以一开始应当检查是否背景 Bitmap 对象已经用一幅图像初始化。如果 Bitmap 对象为空,那么我们知道没有定义背景图像,所以就要使用窗体的 createGraphics 方法来初始化 gdi,以便向窗体绘图。但是,假如 Bitmap 对象非空,就需要用 getGraphics 方法来初始化 gdi 以向 Bitmap 对象绘图。

现在我们有了一个有效的 Graphics 对象,余下的代码就用该 Graphics 对象的方法将信息绘制到启动屏幕上。用 setOpaque 方法的 false 参数会使所有绘制到窗体或 Bitmap 对象上的图形都按透明方式绘制。如果不以这种形式调用此方法,那么 Graphics 对象将在所有文本后面添加灰色背景。每条信息都是用 Graphics 对象的 drawString 方法绘制到屏幕上的。drawString 方法可以定义字符串在绘图平面上绘制的具体位置。因为要在启动屏幕上绘制不同字体大小和风格的字符,所以代码在好几处调用 setFont 来定义 Graphics 对象 gdi 要使用的字体和字型。

回到前面看一看图 8.1 上的产品名称(所显示的“窗体实例”)。为了达到所示文字的三维效果,我们使用了一个简单但有效的技巧——将文字绘制两遍,一遍用深灰色,另一遍用黑色,另外还将灰色文字向右下方稍稍移动。注意在代码中,为了将黑色文字显示在上面,先绘制了灰色文字,然后再绘制黑色文字。在用 Graphics 对象绘图的时候,记住重要的一点:最好在最后绘制想要突出显示的内容。

绘制完成后,代码将验证 Graphics 对象 gdi 是写到窗体上,还是写到 Bitmap 对象上。代码重复了和前面一样的验证过程,验证 Graphics 对象是为哪个对象创建的。当所有绘制是直接用于窗体时,就不需要做任何事情;否则,如果 gdi 对象使用的是 Bitmap 对象,那么就需要显示图片,以将绘制的信息显现出来。这里,Graphics 对象 gdi 被赋予窗体,这样该 Graphics 对象就可以将 Bitmap 对象绘制到窗体上。窗体通过使用所绘图片作为参数之一调用 Graphics 对象的 drawImage 方法来显示该图片。完成后,窗体就会显示绘制上去的所有图像和文本。实际上,如果愿意,还可以使用指定的路径和文件名来调用 Bitmap 对象的 save

方法,将绘制了文本的图片保存在文件中。添加完这段代码后,该启动屏幕实例的类就已设计完成,并可在应用程序中显示。

8.1.7 编写代码以显示启动屏幕

最后一步的工作是编写代码来显示应用程序的启动屏幕。将以下代码添加到 `SplashExample` 的构造函数中,确保将调用后的代码添加到 `initForm()` 后面,以便窗体的初始化过程不会被打断。代码如下:



```
public SplashExample()
{
    // 需要 Visual J++ 窗体设计器的支持
    initForm();

    SplashScreen splash = new SplashScreen(
        "窗体实例",
        "版本 1.0",
        "Copyright 2000",
        "于漫",
        "高手公司",
        "未经许可,不许拷贝");

    splash.setLogoImage("Logo.bmp");
    splash.setSplashBackground("Clouds.bmp");
    splash.showSplash(5000);
}
```

这段代码中,首先应用程序创建了启动屏幕的实例,并使用所有想要显示的字符串信息来调用前面已经自定义的 `SplashScreen` 类构造函数。注意,最后一个传递到构造函数的参数是一个空字符串。这个最后的参数定义了警告文本。如在 8.1.5 小节中提到的,给警告信息赋以空字符串表示通知 `SplashScreen` 对象显示默认的警告文本。调用 `setLogoImage` 和 `setSplashBackground` 分别定义了将图片显示为徽标和显示为背景。因为没有指定路径,所以图片文件必须存放在工程的目录中才能使应用程序正常运行。最后,应用程序以 5000 的值调用 `showSplash`,该项调用将使启动屏幕模式化持续显示 5 秒钟。

到目前为止,已经编写完代码。编译并运行程序,将会看到启动屏幕色彩斑斓的形象。当该程序在系统中正常运行后,试着更改一些设置。首先用标准构造函数取代我们自定义的构造函数创建 `SplashScreen` 的实例,然后调用 `SplashScreen` 的赋值函数来设置显示信息。删除背景图片和徽标图片,看一看启动屏幕上的信息是否依然显示正常。虽然本实例的目的是为了展示如何应用 WFC 显示一个窗体,但其中大多数代码都会其他的程序设计中派上用场。

至此您已经创建了本实例,您也许会对自己说:“是的,这挺不错,但我想添加这些,或者想改改那些……”。不要紧,您尽管改动这些代码,并把它们应用到自己的工程中去。这个实例和全书中其他实例一样都可以修改。试着将以下的功能添加到实例中,以使它更棒一些:

- ◆ 使用注册表检索、存储用户名与公司名。
- ◆ 在启动屏幕上显示已注册使用的所有应用程序的名称(如果合法拥有几套软件的话,那么可以添加这项功能)。甚至,您也许想添加更多的徽标图片,以显示用户注册的每一个产品徽标。这种安排类似于 Visual J++ 和 Visual InterDev 在启动屏幕上显示这两种应用程序符号的方式,只要在用户的系统中这两种产品都得到注册。
- ◆ 添加显示启动屏幕时播放的启动声音。
- ◆ 如果应用程序有许多启动代码,而您并不希望启动屏幕在显示时锁住程序进程,那么请按线程方式创建启动屏幕。在应用程序中也可以不使用设置启动屏幕显示时间的方法,而以进程的方式调用该屏幕,然后在启动代码完成后终止它。
- ◆ 制订一套机制,在启动屏幕上的信息超出原先设置大小时,使启动屏幕自动缩放至适当大小。在本程序中事先设置每条显示文本的位置,可以判断是否文本太大而不能显示,然后在显示启动屏幕以前调整窗体大小,以适应文本的大小。

8.2 创建屏幕保护程序

在前面的实例中,我们创建了一个启动屏幕,以展示如何显示一个窗体。使用启动屏幕更多的是当作画布而不是作为传统的窗体。在下面的实例中,我们将使用窗体的这一功能来创建屏幕保护程序。由于需要具有很大的创造性,并要直接使用操作系统,所以屏幕保护程序是一种非常有意义的应用程序。

要创建 Windows 的屏幕保护程序,需要做很多工作,而不只是以最大化方式显示一个没有边框和标题的窗体。您需要管理命令行、隐藏和显示鼠标指针、处理鼠标和键盘事件等。该实例中创建的屏幕保护程序类似于 Windows 附带的屏幕保护程序。选取框屏幕保护程序和我们的屏幕保护程序一个主要区别是,我们的屏幕保护程序能够显示当前时间。还有一个次要的区别是,Windows 的屏幕保护程序提供了一个用来更改屏幕保护程序属性的对话框,我们创建的屏幕保护程序则需要对其选项进行编码。

通过本节的学习,您将学会从头创建屏幕保护程序的方法。

8.2.1 创建工程及其主窗体

使用 Windows Application 模板创建一个新工程,并将其命名为“DWMarquee”。默认情况下,Visual J++ 创建的窗体将作为屏幕保护程序的显示窗口。将此窗体重新命名为“ScreenSaver.java”,并用 Visual J++ 的 Find And Replace 功能将所有的 Form1 替换为 ScreenSaver。

现在需要设置屏幕保护程序窗体的显示属性,然后添加必要的控件来创建选取框。为窗体添加一个 Label 控件和一个 Timer 控件。因为在显示选取框时要移动控件的位置,所以现在将控件放在何处并不重要。根据表 8.2 设置窗体及其控件的属性。

表 8.2 ScreenSaver.java 组件及其属性设置

组件类型	属性	设置
Form	backColor	Black
	controlBox	false
	borderStyle	None
	keyPreview	true
	text	""
	showInTaskbar	false
	windowState	Maximized
	minimizeBox	false
	maximizeBox	false
	startPosition	Manual
Label	name	lblMarquee
	foreColor	White
	text	""
	visible	false
Timer	name	timerDoSaver
	interval	1
	enabled	true

图 8.6 显示了 ScreenSaver 窗体设计时的样子。

正如在表 8.2 中看到的,屏幕保护程序并没有使用很多控件。Timer 控件能够每隔 1 毫秒执行一次屏幕保护程序和选取框操作,而 Label 控件作为显示的选取框文本。和前面的启动屏幕实例一样,我们需要创建没有边框或标题栏的窗体,以使它看上去像标准的屏幕保护程序窗口。为了全屏幕显示该窗体,将 windowState 属性设置成 Maximized。建议将该屏幕保护程序及创建的所有屏幕保护

程序的 `keyPreview` 属性设置为 `true`。`keyPreview` 属性使窗体能够在具有焦点的控件接收到事件之前,首先接收到所有的键盘事件。虽然我们的屏幕保护程序窗体上没有哪一个控件可以接收焦点,但将 `keyPreview` 属性设置为 `true` 是创建屏幕保护程序时的一个好习惯。



图 8.6 设计时的 `ScreenSaver.java`

8.2.2 将成员变量添加到屏幕保护程序窗体中

在进入屏幕保护程序源代码的核心之前,需要添加一些用来定义屏幕保护程序行为的私有成员变量。之所以将这些成员变量称为私有,是因为我们没有将它们提供给其他程序使用。如果要将这些属性展示给其他用户,您只需提供一些访问私有成员变量的转换方法。将下面的代码添加到 `SplashScreen` 类中:



```
private boolean m_bRandom = true;
private boolean m_bTime = true;
private int m_iSpeed = 20;
private String m_sMarqueeText = new String("Welcome to "
                                             + "Visual J++");

//改变字体
private Font m_fMarqueeFont = new Font("Arial", 62.0f,
                                         FontSize.CHARACTERHEIGHT,
                                         FontWeight.BOLD,
                                         false, false, false);
private Color m_cMarqueeColor = new Color(0x00FFFFFF);
```

```
private int m_iDistance;
private int m_ixStart = 0;
private int m_iyStart = 0;
```

要更改屏幕保护程序的行为,可以更改指定给每一个变量的值。`m_bRandom` 变量指定屏幕保护程序是否在随机的位置显示选取框。`m_bTime` 变量指定是否显示当前时间来替代 `String` 变量 `m_sMarqueeText` 中定义的文本。另外两个成员变量 `m_fMarqueeFont` 和 `m_cMarqueeColor` 用来定义选取框中显示文本的字体和颜色。代码中使用 `m_iDistance`、`m_ixStart` 和 `m_iyStart` 变量来协助定位和显示选取框文本。

8.2.3 将功能添加到屏幕保护程序中(一)

现在,我们已经设计完窗体,并定义了私有成员变量,接下来就可以添加一些代码来使其运行起来。首先,需要创建 `tmrDoSaver` 的 `timer` 事件和 `lblMarquee` 的 `click` 事件的事件处理程序。还需要创建窗体的 `click`、`keyDown` 及 `mouseMove` 事件的处理程序。除了 `tmrDoSaver` 事件处理程序,我们需要创建的所有事件处理程序都将负责关闭屏幕保护程序。根据创建的屏幕保护程序的类型,您也许不希望关闭每一个事件。例如,如果屏幕保护程序同时也是一个游戏,但您希望捕获键盘命令,那么只需将 `click` 事件和 `mouseMove` 事件联系起来。在该实例中,我们将所有的事件都联系起来,以关闭屏幕保护程序。该事件处理程序的代码如下。列出的代码还显示了一个用来完成所有关闭工作的 `Shutdown` 方法。



```
//处理键盘按键事件,关闭屏幕保护程序
private void SSaverKeyDown(Object sender, KeyEvent e)
{
    Shutdown();
}

//处理鼠标移动事件,关闭屏幕保护程序
private void SSaverMouseMove(Object sender, MouseEvent e)
{
    //判断鼠标光标位置之前是否存储过
    if (m_ixStart == 0 && m_iyStart == 0)
    {
        //存储鼠标位置坐标
        m_ixStart = e.x;
        m_iyStart = e.y;
        return;
    }
}
```

```
//判断自屏幕保护程序启动以来,鼠标光标是否移动过
else if (e.x != m_ixStart || e.y != m_iyStart)
    Shutdown();
|

//处理鼠标点击事件,关闭屏幕保护程序
private void ClickMarquee(Object sender, Event e)
|
    Shutdown();
|

//关闭屏幕保护程序,恢复系统
private void Shutdown()
|
    //重新显示光标
    Cursor.show();
    //使计时器无效,从而屏幕保护程序不会继续
    tmrDoSaver.setEnabled(false);
    //结束程序
    Application.exit();
|

//处理 timer 控件事件,执行屏幕保护程序选取框工作
private void DoSaver(Object sender, Event e)
|
    //是否正在显示时间或选取框文本
    if (m_bTime == true)
    |
        //设置选取框标签
        lblMarquee.setText(new Time().formatLongTime());
    |
    else
        //把当前选取框文本设置为选取框标签
        lblMarquee.setText(m_sMarqueeText);
    //设置标签的高度为字体的高度
    lblMarquee.setHeight(
        lblMarquee.getFont().getFontMetrics().height);
    //创建一个图形对象,来决定标签的宽度
    Graphics gdi = this.createGraphics();
    //把标签的字体赋值给图形对象
    gdi.setFont(lblMarquee.getFont());
    //根据图形对象中的文本宽度设置标签的宽度
    lblMarquee.setWidth(gdi.getTextSize(lblMarquee.getText()).x);
    //调用 DoMarquee 方法来做实际的屏保工作
    DoMarquee();
|
```

将上述代码添加到 `ScreenSaver` 类后,需要将这些事件处理程序和它们的事件源联系起来。如将 `ClickMarquee` 事件处理程序与窗体和 `lblMarquee` 的 `click` 事件联系起来。`Visual J++` 允许您将一个组件的事件处理程序与另一个组件的事件源联系起来,只要这些事件创建的事件对象类型相同,并且这些组件都在同一个窗体上。

`SSaverKeyDown` 事件处理程序使屏幕保护程序可以捕获窗体的 `keyDown` 事件。我们是在 `keyPress` 事件上处理 `keyDown` 事件,因为当用户按下任意键时都会触发 `keyDown` 事件,而 `keyPress` 事件不会处理任何键。`SSaverMouseMove` 事件处理程序与窗体的 `mouseMove` 事件联系,使我们可以捕获任意的鼠标移动。

到目前为止,我们讨论的所有事件处理程序都被设计用来在触发事件时关闭屏幕保护程序。所有这些事件处理程序都调用 `Shutdown` 方法来执行关闭屏幕保护程序及维护工作,如重现鼠标指针等。只有 `SSaverMouseMove` 事件处理程序不只是需要调用 `Shutdown` 方法的代码。当一个窗体首次显示时,系统将触发 `mouseMove` 事件。这就造成了一个屏幕保护程序的问题:我们希望捕获 `mouseMove` 事件,以便可以关闭屏幕保护程序,但不是在刚显示屏幕保护程序时。可以用这种办法:在屏幕保护程序启动和退出方法时,`SSaverMouseMove` 事件处理程序代码捕获并存储鼠标的坐标。当再次调用 `SSaverMouseMove` 事件处理程序时,如果鼠标指针的位置与启动屏幕保护程序时的位置不同,那么屏幕保护程序将被关闭。

添加的最后一个事件处理程序是 `DoSaver`。该事件处理程序与 `tmrDoSaver` 的 `timer` 事件联系,并每隔 1 毫秒被调用一次来处理显示选取框的任务。该事件处理程序还包括显示该选取框之前屏幕保护程序要执行的一些预备代码。`DoSaver` 执行一些任务,例如使用当前时间或当前选取框文本来更新标签的显示内容,以及设置选取框 `Label` 控件的大小。该事件处理程序中最有意义的一行是调用 `lblMarquee` 控件的 `setHeight` 方法。该代码使用与控件的 `Font` 对象联系的 `FontMetrics` 对象来确定控件整个文本的顶端高度。`FontMetrics` 类的 `height` 变量返回以像素为单位的高度值,这对于将 `lblMarquee` 的大小调整到适当高度来显示文本是非常有意义的。在代码的最后部分,`DoSaver` 调用了 `DoMarquee` 方法。该方法执行屏幕保护程序的所有基本工作。我们将在该实例的下一步骤中讨论它。

8.2.4 将功能添加到屏幕保护程序中(二)

为了完成屏幕保护程序,需要添加前面提到的 `DoMarquee` 方法,并修改窗体的构造函数和 `main` 方法。首先,将 `DoMarquee` 方法添加到类中。将下面的代码

添加到 SplashScreen 类中:



```
//处理 marquee 的显示和移动
private void DoMarquee()
{
    //获取计算机屏幕的工作区
    Rectangle ssWorkArea = Screen.getWorkingArea(this);
    //如果不是随机显示标签,强制它在垂直方向中对齐
    if (m_bRandom == false)
        //移动标签到新位置
        lblMarquee.setLocation(ssWorkArea.width - m_iDistance,
                                (ssWorkArea.height / 2));
    else
        //否则水平方向添加 marquee 的文本位置,垂直方向位置不动
        lblMarquee.setLocation(ssWorkArea.width - m_iDistance,
                                lblMarquee.getLocation().y);
    //如果标签当前不可见,让它可见
    lblMarquee.setVisible(true);
    //根据用户设置的速度添加标签的距离
    m_iDistance += m_iSpeed;
    //如果标签超出屏幕,则再把它放到正确的位置
    if (lblMarquee.getLocation().x < -(lblMarquee.getWidth()))
    {
        //重设距离为 0
        m_iDistance = 0;
        //判断是否在随机状态
        if (m_bRandom == true)
        {
            //如果标签在顶端,把它移到中间
            if (lblMarquee.getLocation().y == 0)
                lblMarquee.setLocation(
                    lblMarquee.getLocation().x,
                    (ssWorkArea.height/2));
            //如果标签在中间,把它移到底端
            else if (lblMarquee.getLocation().y
                     == ssWorkArea.height / 2)
                lblMarquee.setLocation(
                    lblMarquee.getLocation().x,
                    ssWorkArea.height
                    - lblMarquee.getHeight());
            //重新把标签移到顶端
            else
                lblMarquee.setLocation(
                    lblMarquee.getLocation().x,
                    0);
        }
    }
}
```


DoMarquee 方法显示了选取框的文本。该代码使用 WFC 的 Screen 类来获取它所在的系统的屏幕尺寸。用来确定 lblMarquee 的 Rectangle 对象存储该尺寸。该选取框的主要功能来源于下面的代码段：



```
//如果不是随机显示标签,强制它在垂直方向中
if (m_bRandom == false)
    //移动标签到新位置
    lblMarquee.setLocation(ssWorkArea.width - m_iDistance,
                           (ssWorkArea.height / 2));
else
    //否则水平方向添加 marquee 的文本位置,垂直方向位置不动
    lblMarquee.setLocation(ssWorkArea.width - m_iDistance,
                           lblMarquee.getLocation().y);
```

该代码使用 m_iDistance 成员变量来定位 Label 控件。每次调用该方法时,都会添加 m_iDistance 的值。添加的量取决于 m_iSpeed 成员变量的值,该值定义了选取框文本的显示速度。标签的水平位置取决于屏幕保护程序是被设置为随机显示,还是严格地在屏幕上居中。

为了避免在初始显示屏幕保护程序时,Label 控件位置的混乱,可以在设计时将 Label 控件的 visible 属性设置为 false。在首次调用 DoMarquee 时,使标签再次显现。

此时,您也许会问:“为什么要使用 Label 控件来替代人工绘制文本呢?”原因有两点:首先,它易于使用。它使用简短的代码包含了所有需要的信息。其次,也更为重要的一点,如果通过窗体的 paint 事件绘制文本,常常得到的是闪烁的文本。每次移动文本时重新绘制窗口造成了这种闪烁。无论您选择的是哪一种方法,要完全解决这种“闪烁因素”几乎是不可能的,尤其是以较大的字体显示文本时。但是,使用 Label 控件就有助于减少闪烁。

如果将 m_bRandom 变量设置为 true,那么该方法中最后的 if/else 代码段就会将 Label 控件的垂直位置移动到屏幕的顶端、底部或中间。虽然这种定位并不是完全随机的,但它非常接近于随机,并且减少了创建随机数并验证产生的随机数是否是一个有效的屏幕垂直坐标的麻烦。再次调用 WFC 的 Screen 类以确定屏幕中间及底部的位置。

处理了相对较难的代码后,现在我们将修改 SplashScreen 类的构造函数及 main 方法,以确保屏幕保护程序能够正确显示,并处理 Windows 传送来的命令行参数。ScreenSaver 类的构造函数更改如下。将新代码添加到 initForm() 之后,是

为了它不会影响窗体的初始化过程。



```
public ScreenSaver()
{
    //需要 Visual J++ 窗体设计器的支持
    initForm();

    //设置标签控件的字体和前景色
    lblMarquee.setFont(m_fMarqueeFont);
    lblMarquee.setForeground(m_cMarqueeColor);
    //隐藏鼠标光标
    Cursor.hide();
}
```

添加到构造函数中的代码根据 `m_fMarqueeFont` 和 `m_cMarqueeColor` 中的值来设置 Label 控件的字体和字体颜色。最后一行代码在屏幕保护程序显示时,隐藏鼠标指针。在讨论 Shutdown 方法时曾提到,当关闭屏幕保护程序时,将重新显示鼠标指针。

将下面的代码添加到 ScreenSaver 类的 main 方法中,并完全替换 Visual J++ 生成的默认代码。



```
//显示选项框
if (args[1].startsWith("/c"))
{
    //显示一个消息框使用户知道没有选项,然后退出
    MessageBox.show("There are no options available for "
        + "this screen saver",
        "DWMarquee Screen Saver",
        MessageBox.ICONINFORMATION);
    Application.exit();
}

//重新运行屏幕保护程序
else if (args[1].startsWith("/s"))
{
    //运行屏幕保护程序
    Application.run(new ScreenSaver());
}

//这里不支持密码设置,所以显示一个消息框,然后退出
else if (args[1].startsWith("/a"))
{
    MessageBox.show("Passwords are not available "
        + "for this screen saver",
        "DWMarquee Screen Saver",
        MessageBox.ICONINFORMATION);
}
```

```

        Application.exit();
    }
    //对于任何其他命令行都退出
    else
        Application.exit();

```

添加到屏幕保护程序中的最后一些代码用于处理屏幕保护程序的显示模式。Windows 会将 4 个不同的命令行参数之一传送到屏幕保护程序。这些命令行参数根据用户的操作或屏幕保护程序的显示方式来传送。表 8.3 概述了这些命令行参数,以及导致它们被传送的事件。

为了将实例的重点放在显示屏幕保护程序上,我们没有在屏幕保护程序中添加密码和选项对话框。但是,在下面的实例中我们将添加一个选项对话框。因为没有密码和选项对话框,所以/a 和/c 命令行参数的代码将显示一个消息框,提示用户没有可用的选项或密码,然后退出应用程序。因为我们也没有提供预览监视器的支持,所以当接收到/p 命令行参数时,代码会简单地退出应用程序。/s 命令行参数用于通常的屏幕保护程序显示。

表 8.3 Windows 传送给屏幕保护程序的命令行参数

命令行参数	原因
/s	因为用户停止活动或请求预览屏幕保护程序,所以 Windows 需要显示屏幕保护程序
/a	在 Display 对话框的 Screen Saver 选项卡中选择 Change Password 按钮。该命令行参数可以显示一个密码对话框,使用户可以设置密码
/c	在 Display 对话框的 Screen Saver 选项卡中选择 Settings 按钮。该命令行参数可以显示一个对话框,使用户可以更改屏幕保护程序的设置
/p	在 Display 对话框中选择 Screen Saver 选项卡。该命令行参数在提供的预览监视器中显示屏幕保护程序

8.2.5 编译、打包及发布屏幕保护程序

完成工程的源代码后,我们将编译、打包屏幕保护程序,以便能在 Windows 中看到它的实际情况。

像通常那样编译该工程。成功编译完工程后,下一步是将其封装成可执行程序。从 Project 菜单中选择 DWMarquee Properties,显示 DWMarquee Properties 对话框。查找并选择 Output Format 选项卡,该选项卡允许我们定义工程中创建的包。在本书大多数工程中,我们都将创建一个 Windows.exe 文件。Windows.exe

文件是一个可以像系统中常见的可执行文件那样执行的文件。选中 Enable Packaging 选项,开始创建 .exe 文件。确保选中 Packaging Type 下拉列表中的 Windows EXE 选项,然后输入“DWMarquee.scr”作为文件名。在该工程中我们并没有使用 .exe 作为扩展名,而采用的是 .scr,原因是在搜寻可用的屏幕保护程序时,Windows 将查找该扩展名。保留其他选项的默认设置,然后单击 OK 按钮。

定义完工程的包后,要重新编译该工程,在编译的最后阶段,Visual J++ 将在工程的目录中构建 Windows.exe 文件。要在 Windows 中运行屏幕保护程序,可以将 DWMarquee.scr 文件复制到\Windows\System 目录中(Windows NT 的\Winnt\System32 目录),然后在【控制面板】中打开【显示属性】窗口,从可用的屏幕保护程序中选择该屏幕保护程序。如果单击该对话框中的【更改】或【设置】按钮,将看到我们定义的消息框。如果单击【预览】按钮,将看到我们的屏幕保护程序!

创建完屏幕保护程序后,回到代码并更改其设置,看看屏幕保护程序的表现如何。如设置不同的字体,更改颜色或加快显示速度。

8.3 菜单操作

Microsoft Windows 及其第一代应用程序掀起了一场个人计算的革命。这些最初的图形化用户界面(GUI)标志着纯文本操作系统和应用程序的终结。它们是实现适应当前计算需求的现代应用程序及图形用户界面的重要的第一步。但是,第一代基于 GUI 的应用程序与目前使用的更为完善的用户界面相比,在可用性和吸引力方面都相差很远。虽然最初就可以使用菜单,但是工具栏、状态栏,以及环境菜单等,直到 Windows 3.1 出现之后才被开发出来。事实上,这些附件在 Windows 95 出现之前,都没有被集成到 Windows 中。

在前两节中,我们主要是使用 Forms Designer 来开发各种窗体。讲到的实例为使用 Forms Designer 开发几乎所有类型的窗口或对话框提供了良好的基础。为了增加使用 Forms Designer 的经验,从本节开始介绍如何将菜单、工具栏以及状态栏等添加到应用程序中。这些附件都是常见的类型,并且是几乎所有 Windows 应用程序的重要组成部分。工具栏和环境菜单可以为您的用户定位和访问您的应用程序的功能提供一种简便的方法。在本节和下面的两节中,我们将介绍这些 Windows 开发的重要内容,并演示将这些工具添加到您的应用程序的方法。

本节和下面的两节我们将使用一个程序来演示一步一步将菜单、工具栏和状态栏添加到应用程序的方法。将要开发的这个程序是一种简单但有用的“记事本”式的应用程序。图 8.7 显示了最后的应用程序执行时的外观。



图 8.7 WFCJot 应用程序

在本节和之后的两节不仅要添加窗口附件来改进应用程序,而且还合并上一节中开发的功能——显示启动屏幕。本节的目标是通过使用诸如菜单编辑器和控件编辑器等工具来介绍 Forms Designer 的使用方法。学完本节后,将能够轻松地将菜单添加到您的应用程序中。

8.3.1 创建菜单

菜单是几乎每一个 Windows 应用程序常见的重要功能。可以在 WFC 应用程序中创建两种类型的菜单:主菜单和环境菜单。本节将主要介绍创建主菜单的方法。主菜单是在几乎任何应用程序中都可以看到的传统菜单。通过使用 Visual J++ 菜单编辑器和主菜单组件可以非常容易地将它们添加到应用程序中。菜单编辑器使您能够以图形的方式设计大多数的菜单功能。除了使用菜单编辑器之外,使用 WFC 也可以很方便地通过一些简单的代码在运行时添加和修改菜单。我们不仅会谈到使用菜单编辑器设计菜单的方法,还会涉及到创建菜单的代码。

创建工程和应用程序主窗体

首先创建最初的工程。使用 Empty Project 模板开始一个新工程,然后将其命名为“WFCJot”。该实例将是一个类似于“记事本”的“单文档界面”(SDI)应用程序。将一个新的窗体添加到工程中,然后将其命名为“MainForm.java”。因为该窗体是应用程序的主窗体,所以需要为它添加一些控件,并指定一些属性,以

便它能像“记事本”那样正确地显示。Edit 控件是用户输入文本的地方。MainMenu 控件描绘窗体的主菜单。如果您希望根据用户的操作或应用程序的状态来显示不同的菜单,那么可以将另外的 MainMenu 控件添加到窗体中。在代码中,通过设置包含要切换到的 MainMenu 控件的窗体的 menu 属性,就可以实现在窗体的 MainMenu 控件间切换了。图 8.8 显示了当最初添加这些控件时 Forms Designer 的外观。



图 8.8 没有设置控件属性的设计时的 WFCJot 主窗体

当添加 MainMenu 后,会在窗体的上方出现一个写有“Type Here”的白框。该区域就是菜单编辑器。该窗体会自动将它的 menu 属性设置到您添加的第一个 MainMenu 控件。而将其他 MainMenu 控件添加到窗体中将不会自动更改该窗体的 menu 属性。但是可以在 Properties 窗口中更改该窗体的 menu 属性,或者在代码中设置它。使用表 8.4 设置窗体及其控件的属性。

表 8.4 MainForm.java 组件和属性设置

组件类型	属性	设置
Form	size.x	580
	size.y	500
	startPosition	CenterScreen
	text	WFC Jot
Edit	name	txtDocument
	dock	Fill

续表

组件类型	属性	设置
text	*	
hideSelection	false	
	multiline	true
	acceptsReturn	true
	acceptsTab	true
	scrollBars	Vertical
MainMenu	name	menuMain

图 8.9 显示了正确设置所有属性的设计时的 WFCJot 的主窗体。



图 8.9 正确设置所有属性的设计时的 WFCJot 的主窗体

创建应用程序菜单

现在我们已经创建了一个窗体,并添加了它的控件,下面将创建应用程序的菜单。Visual J++ 的菜单编辑器有两种显示模式。第一种模式是标准的“所见即所得”模式,它允许您指定菜单的标题。第二种模式是菜单名称模式,它允许您在代码中指定菜单对象的名称。通过右击菜单编辑器,然后选择 Edit Names 菜单项,可以显示第二种模式。第一种模式是通常使用的模式,它可以快速地访问菜单和菜单项。而当您要更改与创建的菜单相关的菜单对象名称时,使用第二种模式比较方便。除了使用这两种模式之外,一旦创建了菜单,您就可以使用

Properties 窗口来修改个别的菜单和菜单项。

在菜单编辑器中创建菜单或菜单项是非常容易的。选定 MainMenu 控件,只需单击提供的第一个 Type Here 框,然后输入菜单标题即可。一旦开始键入第一个菜单标题,新的 Type Here 框就会出现在刚刚创建的菜单的右下方。当您添加菜单项时,如果需要在两个现有菜单项间插入一个菜单项,只需右击这两个菜单项中下面的一个,然后在快捷菜单中,单击 Insert New。新的菜单项就会出现在选定菜单项的上面。

一个菜单项与很多属性相关。您可以通过选择来使菜单项具有复选标记,以表示选中该选项,或者在不需要它时,禁用该菜单项。通过从可用的组合键列表中选择一个键值,可以为您的菜单项分配快捷键。快捷键可以方便普通用户,并使那些无法使用鼠标的用户可以访问您的应用程序。可以在运行时使用代码设置除 name(它必须在设计时修改)以外的所有这些属性。

不仅可以使使用 Properties 窗口来添加复选标记或定义菜单快捷键,而且还可以通过菜单编辑器来直接修改它们。要在菜单旁设置或删除复选标记,单击菜单标题或名称左边的空白处(必须处于编辑模式)。要为菜单添加快捷方式,单击菜单标题或名称的右边位置,这时会弹出一个下拉列表,可以为菜单项分配一个字符序号。

要为我们的应用程序创建菜单项,请按照表 8.5 进行设置,该表列出了需要设置的每一个菜单项及其属性。

表 8.5 MainForm.java 菜单设计

标题	名称	父菜单	快捷键	复选标记
&File	mnuFile	None	None	No
&New	mnuNew	mnuFile	Ctrl + N	No
&Open...	mnuOpen	mnuFile	Ctrl + O	No
&Save	mnuSave	mnuFile	Ctrl + S	No
Save&As...	mnuSaveAs	mnuFile	None	No
< Separator >	sepFile1	mnuFile	None	No
E&xit	mnuExit	mnuFile	None	No
&Edit	mnuEdit	None	None	No
Cu&t	mnuCut	mnuEdit	Ctrl + X	No
&Copy	mnuCopy	mnuEdit	Ctrl + C	No
&Paste	mnuPaste	mnuEdit	Ctrl + V	No
&Delete	mnuDelete	mnuEdit	Del	No
< Separator >	sepEdit1	mnuEdit	None	No
&Select All	mnuSelectAll	mnuEdit	Ctrl + A	No
< Separator >	sepEdit2	mnuEdit	None	No

续表

标题	名称	父菜单	快捷键	复选标记
&Find And Replace...	mnuFindReplace	mnuEdit	Ctrl + F	No
&Options	mnuOptions	None	None	No
&Word Wrap	mnuWordWrap	mnuOptions	None	Yes
< Separator >	sepOptions1	mnuOptions	None	No
&Small Font	mnuSmallFont	mnuOptions	None	Yes
&Medium Font	mnuMediumFont	mnuOptions	None	No
&Large Font	mnuLargeFont	mnuOptions	None	No
&Help	mnuHelp	None	None	No
&About WFC Jot...	mnuAbout	mnuHelp	None	No

完成创建菜单的工作后,它们的外观应该如图 8.10 所示。

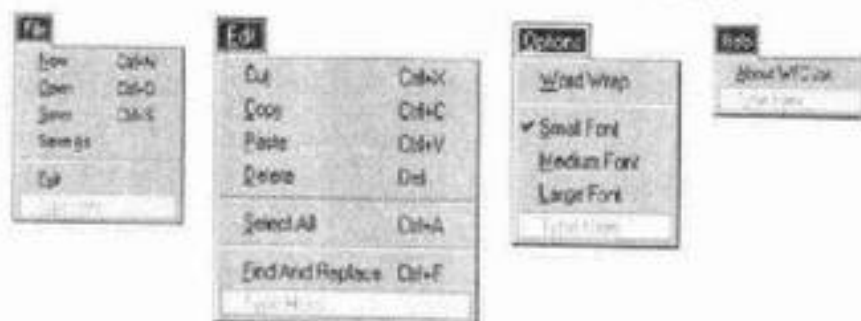


图 8.10 开发期间的 WFCJot 的菜单

除了 Option 菜单外,创建的大多数菜单的功能都是很常见的。选中/取消 Word Wrap 菜单项,使用户可以决定是否希望出现文字回行情况。我们将在默认情况下保持选中该菜单项,因为这是较为常见的情况。Small Font、Medium Font 和 Large Font 菜单项只能选中其中一个,它们使用户可以定义在 txtDocument 控件中使用的字体大小。

现在已经创建了 MainForm 菜单,保存您所做的工作,以便将菜单代码添加到类中,然后看一看菜单编辑器添加到窗体以创建菜单的所有代码。以下是精简的代码片断。



```
MainMenu mnuMain = new MainMenu();
MenuItem mnuFile = new MenuItem();
MenuItem mnuNew = new MenuItem();
MenuItem mnuOpen = new MenuItem();
MenuItem mnuSave = new MenuItem();
MenuItem mnuSaveAs = new MenuItem();
```

```
MenuItem sepFile1 = new MenuItem();
MenuItem mnuExit = new MenuItem();

mnuNew.setShortcut(Shortcut.CTRL_N);
mnuNew.setText("&New");

mnuOpen.setShortcut(Shortcut.CTRL_O);
mnuOpen.setText("&Open");

mnuSave.setShortcut(Shortcut.CTRL_S);
mnuSave.setText("&Save");

mnuSaveAs.setText("Save &As...");

sepFile1.setText("-");

mnuExit.setText("E&xit");

mnuFile.setMenuItems(new MenuItem[] {
    mnuNew,
    mnuOpen,
    mnuSave,
    mnuSaveAs,
    sepFile1,
    mnuExit});

mnuFile.setText("&File");

mnuMain.setMenuItems(new MenuItem[] {
    mnuFile,
    mnuEdit,
    mnuOptions,
    mnuHelp});
```

在以上显示的创建 File 菜单的代码中,您会注意到每一个菜单项都是一个对象。这种安排非常有效,因为它允许您在运行时通过将一个菜单项赋给另一个菜单项来更改整个菜单的布局。该代码首先创建 MainMenu 对象及在 Forms Designer 创建的每个菜单的 MenuItem 对象。

在 initForm 方法中,setText 方法指定了每个 MenuItem 对象的标题。请注意,我们在 SaveAs 菜单项后创建的分隔符是用“-”来表示的。当 Visual J++ 看到“-”,它会在相应的位置插入适当的分隔符。如果 MenuItem 对象指定了快捷方

式,那么 `initForm` 方法将使用 `Shortcut` 类中的静态成员变量来调用 `setShortcut`。`Shortcut` 类定义用来表示可以分配给菜单项的快捷键组合的静态成员变量。

`initForm` 初始化 `MenuItem` 对象以后,代码使用 `setMenuItems` 方法将 `MenuItem` 对象分配给 `mnuFile` 菜单。`setMenuItems` 方法接受 `MenuItem` 数组作为参数。该数组在调用具有分配的 `MenuItem` 对象名称的方法中被创建。代码然后进行相同的 `setMenuItems` 调用,但这次是从 `mnuMain MainMenu` 对象进行的,将 `mnuFile` 菜单指定为应用程序的主菜单的一个菜单项。

当您查看使用菜单编辑器创建的代码时,很容易发现在运行时使用代码添加和创建菜单项的方法。要将 `Close` 菜单项添加到 `mnuFile` 中,只需创建一个 `MenuItem` 对象,为它分配文本 `Close`,然后使用 `MenuItem` 类的 `add` 方法将它分配给 `mnuFile`。要在 `Save As` 菜单项后插入新的菜单项,可以从 `mnuFile` 菜单对象进行 `insert` 方法的相关调用,并在 `File` 菜单中选择一个指定的位置来插入新的菜单项。下面的代码片断显示了两种将菜单项添加到现有菜单的方法。



```
MenuItem mnuFile = new MenuItem();
MenuItem mnuClose = new MenuItem();

private void initForm()
{
    mnuClose.setText("&Close");
    mnuFile.add(mnuClose);

    // or
    // mnuFile.insert(4, mnuClose);
}
```

在 `Visual J++` 中创建菜单不仅简单,而且非常灵活。只需使用菜单编辑器就可以创建整个菜单系统。如果需要进一步调整菜单创建的过程,那么您可以通过添加几行代码来添加新的菜单和菜单项。

现在已经定义了菜单,下面只需要将一些私有成员变量添加到 `MainForm.java` 中,来提供关于应用程序状态的信息。将下面的代码添加到 `MainForm.java` 类正文的上面。



```
//私有成员变量
private String m_sAppName = new String("WFC Jot");
private String m_sFilename = new String("Untitled");
private boolean m_bDirty = false;
private boolean m_bWordWrap = true;
private static final int SMALLFONT = 0;
```

```
private static final int MEDIUMFONT = 1;
private static final int LARGEFONT = 2;
private int m_iCurrentFont = SMALLFONT;
```

`m_sAppName` 成员变量用来保存应用程序的名称,并且将在应用程序不同的地方使用它。在下一小节中,当添加代码来打开和保存文件时,将使用 `m_sFilename` 和 `m_bDirty`。我们将使用 `m_sFilename` 来存储当前的文件名或 `Untitled` (如果硬盘中不存在该文件)。使用 `m_bDirty` 成员变量作为标记来决定 `txtDocument` 中的文本是否被更改。在下一小节中,将添加使用 `m_bWordWrap` 和 `m_iCurrentFont` 成员变量来设置 `Word Wrap` 的选中状态和字体大小菜单项的代码。

将启动屏幕添加到应用程序中

现在利用上一节学过的创建启动屏幕的方法,添加本应用程序的启动屏幕。该启动屏幕会使应用程序看起来很专业,并且它将演示制作启动屏幕窗体是多么方便。要将启动屏幕添加到工程中,可以从 `Project` 菜单中选择 `Add Item`,显示 `Add Item` 对话框。选择 `Existing` 选项卡,定位到 `SplashExample` 工程的位置。双击 `SplashScreen.java` 文件,将启动屏幕窗体添加到工程目录中。将下面的代码添加到 `MainForm` 构造函数 `initForm` 调用的后面,如下所示:



```
public MainForm()
{
    super();
    initForm();

    //创建启动屏幕
    SplashScreen splash= new SplashScreen (m_sAppName,
                                           "Version 1.0",
                                           "Copyright (c)1998",
                                           "Unknown User",
                                           "Unknown Company",
                                           "");

    splash.setLogoImage("WFCJotLogo.bmp");
    splash.showSplash(5000);
}
```

如果您使用过上一节中的实例,那么应该熟悉这些代码。代码简单地创建了一个启动屏幕实例,分配为应用程序指定的名称及其他启动屏幕信息,然后调用 `ShowSplash` 方法来显示启动屏幕 5 秒钟。在本书附带的光盘中包括了一个徽标 `WFCJotLogo.bmp`,可以将其复制到 `WFC Jot` 的工程文件夹中,供启动屏幕之用。

如果编译并运行该工程,那么启动屏幕将显示 5 秒钟,然后显示 WFC Jot 的主窗体。该窗体没有什么功能,只有一些菜单项可以用来查看。查看完应用程序,继续下一小节,下一小节将介绍如何将事件处理程序与菜单联系起来,以及添加 WFC Jot 应用程序的基础功能。

8.3.2 创建菜单事件处理程序

通过上一小节的操作,可以发现 Visual J++ 确实具有灵活的菜单结构,以及易于使用的菜单编辑器。这种灵活性和易用性可以扩展到为使用菜单编辑器设计的菜单创建事件处理程序的过程中。创建菜单事件处理程序的过程与编写其他事件源的事件的过程相同。在本小节中,我们将添加前面章节中定义的菜单的事件处理程序。还要将支持代码添加到菜单事件处理程序中,以便使 WFC Jot 成为一个有用的应用程序。

添加 WFC Jot 的菜单事件处理程序

这里将使用本书附带的光盘 Samples \chap08 \WFCJot2 目录下的 WFC Jot 工程。

我们具有一些为 WFC Jot 创建的事件处理程序,并将一些菜单联系到一个事件处理程序中。使用表 8.6 来定义菜单事件处理程序。其中大多数事件处理程序都使用默认的菜单事件处理程序名称(< MenuObject > _ < Event Name >)。

表 8.6 MainForm.java 菜单事件处理程序

菜单项	事件	事件处理程序名称
mnuNew	click	mnuNew_click
mnuOpen	click	mnuOpen_click
mnuSave	click	mnuSave_click
mnuSaveAs	click	mnuSaveAs_click
mnuExit	click	mnuExit_click
mnuEdit	popup	InitEdit
mnuCut	click	mnuCut_click
mnuCopy	click	mnuCopy_click
mnuPaste	click	mnuPaste_click
mnuDelete	click	mnuDelete_click
mnuSelectAll	click	mnuSelectAll_click
mnuFindReplace	click	mnuFindReplace_click

续表

菜单项	事件	事件处理程序名称
mnuOptions	popup	InitOptions
mnuWordWrap	click	mnuWordWrap_click
mnuSmallFont	click	FontSizeChange
mnuMediumFont	click	FontSizeChange
mnuLargeFont	click	FontSizeChange
mnuAbout	click	mnuAbout_click

WFC Jot 菜单中的大多数菜单项都有一个相应的 click 事件菜单处理程序。例外的是 mnuSmall、mnuMedium 和 mnuLarge 菜单项,它们都对应于 FontSizeChange 事件处理程序。我们只使用一个事件处理程序是因为我们希望在一个地方处理所有的字体大小的更改。为其中的一个菜单项创建事件处理程序后,从事件旁提供的下拉列表中选择菜单。InitOptions 和 InitEdit 方法对于其他事件处理程序的规则也有所不同。这些处理程序与它们菜单的 popup 事件相关联。当用户单击一个菜单,但在出现菜单项之前,发生 popup 事件。我们将在下一小节中使用 InitEdit 和 InitOptions 事件处理程序来决定启用或选中 Edit 和 Options 菜单中的那个菜单项。

将支持代码添加到 File 菜单的事件处理程序中

现在我们已经创建了菜单处理程序,下面将添加支持代码。我们不是一次添加所有的代码,而是添加该步骤及接下来的两步中涉及到的代码。这里将添加 File 菜单项的事件处理程序代码。另外还将添加一些帮助程序的方法,并对 MainForm 的代码进行一些小的更改。首先,将下面的代码添加到 MainForm.java 源文件的前面和其他导入代码的后面:

```
import com.ms.wfc.io.File;
```

现在,添加下面的代码来支持 File 菜单。



```
private void mnuNew_click(Object source, Event e)
{
    //检查是否已有文件打开
    if(HandleDir(source, e) == true)
    {
        txtDocument.setText("");
        m_sFilename = "Untitled";
        ChangeDialogCaption();
        m_bDirty = false;
    }
}
```

```

private void mnuOpen_click(Object source, Event e)
{
    if (HandleDirL(source, e) == false)
        return;
    else
    {
        OpenFileDialog ofd = new OpenFileDialog();
        //设置打开对话框中的文件筛选器
        ofd.setFilter("Text Files (*.txt)|*.txt"
            + " All Files (*.*)|*.*");
        //设置选项确保文件路径有效和文件存在隐藏对话框中的只读选项
        ofd.setCheckPathExists(true);
        ofd.setCheckFileExists(true);
        ofd.setShowReadOnly(false);
        //设置初始文件名为*.txt
        ofd.setFileName("*.txt");
        //如果 Cancel 键未被单击,显示 OpenFileDialog 对话框
        if (ofd.ShowDialog() == DialogResult.OK)
        {
            //检查用户打开有效的文件类型
            try
            {
                //打开 OpenFileDialog 对话框指定的文件
                File tempFile = File.Open(ofd.GetFileName());
                //存储文件的长度
                long fileSize = tempFile.GetLength();
                //载入带有文件内容的 edit 控件
                txtDocument.setText(
                    tempFile.ReadStringCharsAnsi((int)fileSize));
                //关闭文件对象
                tempFile.close();
                //设置当前文件名为打开的文件名
                m_sFilename = ofd.GetFileName();
                //改变对话框的标题属性以反映文件名的变化
                ChangeDialogCaption();
                //重置旧文件标志
                m_bDirty = false;
            }
            catch (Exception ex)
            {
                /* * 如果出现异常,那么这个异常通常是因为
                * 打开一个无效的文本文件 */
            }
        }
    }
}

```



```

        MessageBox.show("The file that you are opening"
            + "is not a valid text file.",
            m_sAppName,
            MessageBox.ICONERROR);
        //确认文件被清空
        txtDocument.setText("");
    }
}

private void mnuSave_click(Object source, Event e)
{
    SaveFile(source, e);
}

private void mnuSaveAs_click(Object source, Event e)
{
    SaveFileAs(source, e);
}

private void mnuExit_click(Object source, Event e)
{
    //检查旧文件标志已被设置
    if(HandleDirt(source, e) == true){
        //退出应用程序
        Application.exit();
    }
}

/* * 判断文件是旧文件还是新文件
 * 如果是旧文件则存盘
 * 如果当前文件能被关闭则返回一个布尔量 */
private boolean HandleDirt(Object sender, Event e)
{
    boolean returnVal = false;

    //判断文件是否为旧文件
    if (m_bDirty == false)
        return true;
    //询问用户是否存储文件
    int answer = MessageBox.show("Do you wish to save changes"
        + "to" + m_sFilename + "?",
        m_sAppName,
        MessageBox.ICONQUESTION
        + MessageBox.YESNOCANCEL);

    switch (answer)

```

```

        |
        case DialogResult.YES:
            //把文件存到硬盘
            returnVal = SaveFile(sender, e);
            break;
        case DialogResult.NO:
            //是,会破坏活动文档
            returnVal = true;
            break;
        case DialogResult.CANCEL:
            //否,不会破坏活动文档
            returnVal = false;
            break;
        |
        return returnVal;
    |

//把活动文档存成文件
private boolean SaveFile (Object sender, Event e)
|
    boolean returnVal = false;

    //检查是否有一个存在的文件名
    if (m_sFilename == "Untitled")
        // 既然没有文件名,显示 SaveAs 对话框并创建一个文件名
        returnVal = SaveFileAs(sender, e);
    else
    |
        //创建一个文件对象和文件
        File tempFile = File.create(m_sFilename);
        //把 edit 框中的内容写到文件
        tempFile.writeChars(txtDocument.getText());
        //关闭文件
        tempFile.close();
        //重置旧文件标志
        m_bDirty = false;
        //返回操作成功
        returnVal = true;
    |
    //返回文件存储的状态
    return returnVal;
|

//显示 SaveFile 对话框以存盘
private boolean SaveFileAs(Object sender, Event e)

```

```

        boolean returnVal = false;
        SaveFileDialog fsd = new SaveFileDialog();

        // 设置显示在对话框中的初始文件名
        fsd.setFileName(m_sFilename);
        // 设置对话框中的筛选器
        fsd.setFilter("Text Files (*.txt)|*.txt"
            + "|All Files (*.*)|*.*");
        // 设置选项以使文件路径有效
        fsd.setCheckPathExists(true);
        // 显示 FileSave 对话框。如果 Cancel 键未被单击,则继续
        if (fsd.ShowDialog() == DialogResult.OK)
        {
            // 从对话框获得文件名
            String tempString = fsd.GetFileName();
            // 通过寻找 "." 检查文件扩展名
            if (tempString.indexOf(".") == -1)
            {
                // 如果没有文件扩展名,则匹配以 ".txt"
                tempString = tempString + ".txt";
                // 指定扩展名到文件对话框的返回值
                fsd.setFileName(tempString);
            }

            // 在对话框中文件名的基础上创建一个文件
            File tempFile = File.create(fsd.GetFileName());
            // 把 edit 控件中的内容写到文件中
            tempFile.writeStringCharsAnsi(txtDocument.getText());
            // 关闭文件
            tempFile.close();
            // 为在对话框中指定的文件设置活动文件名
            m_sFilename = fsd.GetFileName();
            // 改变对话框标题以反映文件名的变化
            ChangeDialogCaption();
            // 设置旧文件标志
            m_bDirty = false;
            returnVal = true;
        }

        // 返回操作成功
        return returnVal;
    }

    // 改变对话框标题以反映当前打开的文档
    private void ChangeDialogCaption()
    {

```

```
//设置窗体的标题以显示应用程序名和当前打开的文件名
this.setText(m_sAppName + " - " + m_sFilename);
```

请注意,我们创建的所有事件处理程序都具有相同的参数。在 WFC 中,所有的菜单事件都被相同对待。当查看事件处理程序的帮助程序函数时,记住这一点是非常重要的。

引入到代码中的第一个事件处理程序是 `mnuExit_click`。该代码决定用户是否通过调用帮助程序的 `HandleDirt` 方法来更改当前打开的文档。如果该文档不是旧的(dirty),那么它将关闭应用程序。`HandleDirt` 是一个事件处理程序,除了 `Boolean` 类型的返回值之外,该方法的信号与其他菜单事件处理程序都是一致的。我们使这个帮助程序方法接受相同的参数作为菜单事件处理程序,这是因为 `HandleDirt` 隐含有将文档保存到文件中的事件处理程序。我们需要一些参数信息来调用这些事件处理程序。

即使不需要从 `HandleDirt` 调用任何菜单事件处理程序,仍应该传递事件的参数。`HandleDirt` 的代码检查私有成员变量 `m_bDirty`,来查看用户是否更改了 `txtDocument` 中的文本。如果 `m_bDirty` 为真,那么该代码将显示一个允许用户保存对文本所做的更改的消息框。用户可以选择 Yes、No 或 Cancel。如果用户选择的是 Yes,那么将调用具有 `HandleDirt` 参数的 `SaveFile`,`SaveFile` 类似于 `HandleDirt`,其中具有事件处理程序的参数列表;如果在消息框中选择的是 No,那么代码只需将局部 `boolean` 变量 `returnVal` 设置为 `true`,这个局部变量被分配到方法的返回值;如果用户选择的是 Cancel,那么 `returnVal` 将被设置为 `false`,`HandleDirt` 的 `boolean` 返回值用来通知调用程序,保留 `txtDocument` 的现有内容。

上一小节中提到的 `SaveFile` 帮助程序方法也被 `mnuSaveFile_click` 事件处理程序调用。`SaveFile` 代码的开头声明将要在代码中设置、并被作为方法返回值分配的局部 `Boolean` 变量。声明后,代码将检查私有成员变量 `m_sFilename` 的值,来查看当前文件是已经存在于硬盘上,还是尚未指定标题。如果文件还没有保存在硬盘上,那么代码将继续调用帮助程序方法 `SaveFileAs`。否则,代码将使用当前文件的名称创建一个 `File` 类实例,并使用 `writeStringCharsAnsi` 调用将 `txtDocument` 的内容写到文件中。然后代码重新设置 `m_bDirty` 标记,因为文档刚被保存过,所以将返回值 `true`。

`SaveFileAs` 类似于 `SaveFile` 帮助程序方法,并且它是从 `mnuSaveAs_click` 菜单事件处理程序及 `SaveFile` 调用的。`SaveFile` 和 `SaveFileAs` 代码之间的主要区别是 `SaveFileAs` 显示 `SaveFileDialog` 对话框,使用户可以选择文件名和位置来保存文件。一旦完成写入用户指定文件的操作,`SaveFileAs` 就调用我们添加到代码中的帮助程序最后一个方法 `ChangeDialogCaption`。该帮助程序方法只更改窗体的标题来反

映当前打开的文档的名称。除了从 SaveFileAs 调用之外, ChangeDialogCaption 方法还会从 mnuOpen_click 和 mnuNew_click 事件中处理程序, 以及 MainForm 构造函数调用。

我们添加的最后两个事件处理程序是 mnuOpen_click 和 mnuNew_click。mnuNew_click 事件处理程序调用 HandleDirt 来查看在继续之前它是否需要保存现有的文件。如果 HandleDirt 返回 true, 那么代码将清除 txtDocument 的内容, 重新将当前文件名设置为 Untitled, 然后调用 ChangeDialogCaption 来重新设置 MainForm 的标题。与 mnuSave_click 和 mnuSaveAs_click 不同, mnuOpen_click 执行它的大多数工作来代替调用大量帮助程序方法。代码从检查文档是否为“旧”(即是否存在该文档)开始。如果该文档不存在, 那么代码将创建 OpenFileDialog 的实例, 设置它的筛选和默认的文件名, 并显示对话框。如果用户没有在 OpenFileDialog 对话框中选择 Cancel, 那么代码将继续创建 File 对象, 然后将整个文件内容读入到 txtDocument 中。我们将打开文件和读取内容的代码放在 try/catch 块中。mnuOpen_click 方法, 像 mnuSave_click 和 mnuSaveAs_click 那样, 调用帮助程序方法 ChangeCaptionDialog 来重新设置 MainForm 的标题, 以反映当前打开的文件。

必须进行的另一个更改是在 MainForm 的构造函数中添加 ChangeDialogCaption 调用。该调用确保当首先打开窗体时能够正确显示 MainForm 标题。

我们通过 File 菜单的练习进行了处理事件和创建帮助程序方法的操作。使用帮助程序方法很重要, 因为它使代码更加模块化, 并可被其他事件重复使用。

添加 Edit 菜单事件处理程序的支持代码

Edit 菜单是接下来要添加事件处理程序代码的地方。与 File 菜单的代码相比, Edit 菜单的代码较短。我们还将添加 txtDocument 的 textChanged 事件的处理程序。该事件处理程序将允许我们在用户更改 txtDocument 时设置旧标记。在第一步创建的事件处理程序和 textchanged 事件处理程序的 MainForm.java 中添加下面的代码。



```
private void txtDocument_textChanged(Object source, Event e)
{
    //设置标志来判断文档中的变化
    m_bDirty = true;
}

private void mnuFindReplace_click(Object source, Event e)
{
    //TODO IN LATER EXAMPLES
}
```

```
private void mnuSelectAll_click(Object source, Event e)
{
    //全选 edit 控件中的文本
    txtDocument.selectAll();
}

private void mnuDelete_click(Object source, Event e)
{
    //清除选中的当前文本
    txtDocument.setSelectedText("");
}

private void mnuPaste_click(Object source, Event e)
{
    //创建一个数据对象来存储剪贴板的内容
    DataObject tempData = (DataObject) Clipboard.getDataObject();
    //从剪贴板检索出数据
    try
    {
        txtDocument.setSelectedText(
            (String)tempData.getData(String.class));
    }
    catch (Exception ex)
    {
        MessageBox.show("The current contents of the"
            + " Clipboard are not in text format"
            + "...paste operation canceled.",
            m_sAppName,
            MessageBox.ICONEXCLAMATION);
    }
}

private void mnuCopy_click(Object source, Event e)
{
    //把当前选中的文本发送到剪贴板
    Clipboard.setDataObject(txtDocument.getSelectedText());
}

private void mnuCut_click(Object source, Event e)
{
    //把当前选中的文本发送到剪贴板
    Clipboard.setDataObject(txtDocument.getSelectedText());
    //从 edit 控件中移出当前选中的文本
    txtDocument.setSelectedText("");
}

private void InitEdit(Object source, Event e)
{
    //根据选中的文本判断剪切、拷贝和删除菜单项的状态
    if (txtDocument.getSelectedText().length() == 0)
```

```

        mnuCut.setEnabled(false);
        mnuCopy.setEnabled(false);
        mnuDelete.setEnabled(false);
    }
    else
    {
        mnuCut.setEnabled(true);
        mnuCopy.setEnabled(true);
        mnuDelete.setEnabled(true);
    }
    //在剪贴板中文本的有效性基础上设置粘贴按钮命令
    try
    {
        mnuPaste.setEnabled(Clipboard.getDataObject()
                            .getDataPresent(String.class));
    }
    catch (Exception ex)
    {
        mnuPaste.setEnabled(false);
    }
}

```

现在需要将事件处理程序 `txtDocument.textChanged` 联系到 `txtDocument` 的 `textChanged` 事件上。用户对文档进行的任何更改都会触发 `txtDocument` 的 `textChanged` 事件,然后事件处理程序将 `m_bDirty` 标记设置为 `true`。

我们应该检查的第一个事件处理程序是 `InitEdit`。它是当用户单击 `mnuEdit` 时,在 Windows 显示任何 `mnuEdit` 的菜单项之前被 `popup` 事件调用的事件处理程序。`InitEdit` 的代码查看 `txtDocument` 中是否有选定的文本。如果当前没有选定的文本,那么代码将禁用 `mnuCut`、`mnuCopy` 和 `mnuDelete` 菜单项,因为没有选定的文本用来进行剪切、复制或删除操作。代码还将查看【剪贴板】中是否有要被粘贴的文本。如果没有,那么代码将禁用 `mnuPaste`。该过程只需要一行代码。首先,调用将进行存储【剪贴板】内容的 `DataObject` 类的实例。`DataObject` 实例可以存储对于【剪贴板】有意义的信息。一旦创建了对象,调用就会立即进行 `DataObject` 的 `getDataPresent` 方法。该方法接受类的类型作为参数,并且如果存储在 `DataObject` 实例中的数据与该参数相同,那么它将返回一个 `Boolean` 值。因为检查的是文本,所以传递的 `getDataPresent` 为 `String.class` 类型,根据它的返回值,代码将禁用或启用 `mnuPaste` 菜单项。

因为刚刚讨论了 `DataObject` 类和【剪贴板】,所以下面来看一看 `mnuPaste.click`。该事件处理程序开始的代码创建了 `DataObject` 的实例,并将【剪贴板】的内容传递给 `DataObject`。然后代码尝试着将【剪贴板】的内容分配给 `txtDocument`。


```
        m_iCurrentFont = SMALLFONT;
    }
    else if (mnuMediumFont == checkingMenu)
    {
        //设置字体大小为 16
        txtDocument.setFont(new Font(txtDocument.getFont(),
                                    16.0f,
                                    FontSize.CHARACTERHEIGHT));
        m_iCurrentFont = MEDIUMFONT;
    }
    else
    {
        //设置字体大小为 24
        txtDocument.setFont(new Font(txtDocument.getFont(),
                                    24.0f,
                                    FontSize.CHARACTERHEIGHT));
        m_iCurrentFont = LARGEFONT;
    }
}

private void InitOptions(Object source, Event e)
{
    //更新选项菜单中的 Word Wrap 项
    mnuWordWrap.setChecked(m_bWordWrap);

    //清除选项菜单中的所有标记
    mnuSmallFont.setChecked(false);
    mnuMediumFont.setChecked(false);
    mnuLargeFont.setChecked(false);

    //检查确切的字体大小对象
    switch(m_iCurrentFont)
    {
    case SMALLFONT:
        mnuSmallFont.setChecked(true);
        break;
    case MEDIUMFONT:
        mnuMediumFont.setChecked(true);
        break;
    default:
        mnuLargeFont.setChecked(true);
    }
}

private void mnuAbout_click(Object source, Event e)
{
    //TODO IN LATER EXAMPLES
}
```

`mnuWordWrap_click` 事件处理程序负责设置 `txtDocument` 的文字回行状态。代码首先为 `m_bWordWrap` 成员变量分配与 `txtDocument` 的 `wordWrap` 属性相反的值。然后使用该 `Boolean` 值来设置 `txtDocument` `wordWrap` 属性新的状态,并决定为 `txtDocument` 设置的滚动条类型:当文字回行有效时设置垂直滚动条;当禁用文字回行时,同时设置水平和垂直滚动条。`FontSizeChange` 事件处理程序与 3 个字体大小菜单相关联,并负责设置 `txtDocument` 的字体。事件处理程序的代码从创建用户选定的 `MenuItem` 对象的引用开始。代码使用该引用来决定用户选中的是 3 个菜单项中的哪一个,然后相应地设置 `txtDocument` 的字体。代码还设置 `m_iCurrentFont` 成员变量来反映用户选定的内容。因为 `Font` 对象是不可变的,所以 `txtDocument` 的 `setFont` 方法的调用将根据 `txtDocument` 当前的字体、字体大小来创建一个新的 `Font` 对象。可以使用 `Font` 类的各种构造函数来更改现有的字体。

`Options` 菜单 `popup` 事件调用的 `InitOptions` 事件处理程序用来设置 `Word Wrap` 的选中状态及字体大小菜单项。`InitOptions` 首先使用 `m_bWordWrap` 成员变量设置 `mnuWordWrap` 的选中状态,然后代码取消选中所有字体大小菜单项。`InitOptions` 使用 `switch` 语句和 `m_iCurrentFont` 成员变量来设置字体大小菜单项的选中状态,以使与 `txtDocument` 当前字体大小相应的菜单项出现复选标记。

最后,只有 `mnuAbout_click` 事件处理程序保持空白。该处理程序和 `mnuFind-Replace_click` 将在下一小节中讨论。在该处理程序中,我们将创建一个简单的 `About` 框,但不影响程序的主要功能。现在,可以编译、生成并运行 `WFC Jot` 了。您将会看到,这个简单的小应用程序可以做很多事情,并且相当有用。试一试这个应用程序,然后继续下一小节,您将会通过为应用程序创建环境菜单来继续构建 `WFC Jot`,以更加完善它。

8.3.3 创建环境菜单

如果您想在 `Windows 95` 或 `Windows NT` 下更方便地操作,那么应该考虑将环境菜单(右击出现的快捷菜单)添加到应用程序中。环境菜单通常是现有菜单的子集,它在应用程序内的重要位置提供有用的菜单命令。在应用程序内或应用程序与操作系统之间使用鼠标右键执行操作时,快捷菜单将为用户提供多个命令选项。

在本小节中,我们将把环境菜单支持添加到 `WFC Jot` 应用程序中。将讨论如何使用 `Forms Designer` 或代码来创建环境菜单。我们创建的菜单将在用户右击 `txtDocument` 的 `Edit` 控件时显示。该环境菜单组合了前面章节中创建的各种菜单命令。完成本节后,您将学会将环境菜单添加到应用程序中。

将 ContextMenu 控件添加到工程中

可以使用本书附带光盘中的代码来完成这个源代码。打开 Samples \chap08 \WFCJot3 目录下的 WFCJot 工程。要添加的环境菜单由 Edit 菜单的菜单项和 Options 菜单的字体大小菜单项组成。要开始这项操作,可以将 ContextMenu 控件添加到 MainForm.java 中,然后将该控件命名为“cmnDocument”。因为当用户右击 txtDocument 的 Edit 控件时将显示该环境菜单,所以需要将 cmnDocument 与 txtDocument 关联起来。要将环境菜单与 txtDocument 关联起来,可以使用 Properties 窗口的下拉列表将 txtDocument 的 contextMenu 属性设置为 cmnDocument。然后,创建 cmnDocument popup 事件的处理程序,并将其命名为 InitContext。在下一步骤中,我们将 MenuItem 添加到为 txtDocument 显示的 cmnDocument。



注意 您也许会问“当用户右击 Edit 控件时,环境菜单如何显示呢?”Visual J++ 会自动为您处理这件事情。您所要做的就是创建一个 ContextMenu 对象,然后使用该控件的 contextMenu 属性将其分配给一个控件。于是,只有当 ContextMenu 控件具有分配给定的 MenuItem 对象时,才会显示环境菜单。

为环境菜单分配菜单项

可以通过两种方式将菜单分配给 ContextMenu 控件:可视地使用菜单编辑器;使用代码。如果您正在创建独立于 MainMenu 控件中定义的菜单的菜单项,那么使用菜单编辑器将菜单项添加到 ContextMenu 控件是很有用的。要编辑 ContextMenu 控件的菜单项,只需在它的窗体中双击 ContextMenu 控件。与在菜单编辑器中编辑主菜单不同,环境菜单没有顶级菜单项。取而代之的是,环境菜单的名称将显示在最上层。除此之外,使用菜单编辑器创建环境菜单项是相同的。

出于我们的目的,我们使用代码来代替使用菜单编辑器,来将菜单项添加到 ContextMenu 控件中,这样可以使使用现有的菜单项,而不用重新创建。将下面的代码添加到 MainForm 类定义的开始处、其他私有成员变量声明之后。



```
//Context menu items
private MenuItem cmnCut;
private MenuItem cmnCopy;
private MenuItem cmnPaste;
private MenuItem cmnSelectAll;
private MenuItem cmnSmallFont;
private MenuItem cmnMediumFont;
private MenuItem cmnLargeFont;
```

将下面的代码添加到 MainForm 构造函数现有代码的后面:



```
//克隆剪切、拷贝、粘贴和全选菜单项
cmnCut = mnuCut.cloneMenu();
cmnCopy = mnuCopy.cloneMenu();
cmnPaste = mnuPaste.cloneMenu();
cmnSelectAll = mnuSelectAll.cloneMenu();
//克隆字体大小菜单项
cmnSmallFont = mnuSmallFont.cloneMenu();
cmnMediumFont = mnuMediumFont.cloneMenu();
cmnLargeFont = mnuLargeFont.cloneMenu();

//添加剪切、拷贝、粘贴和全选菜单项副本到环境菜单
cmnDocument.add(cmnCut);
cmnDocument.add(cmnCopy);
cmnDocument.add(cmnPaste);
cmnDocument.add(cmnSelectAll);
//添加菜单分隔符
cmnDocument.add(new MenuItem("-"));
//添加字体大小菜单项副本到环境菜单
cmnDocument.add(cmnSmallFont);
cmnDocument.add(cmnMediumFont);
cmnDocument.add(cmnLargeFont);
```

前3行代码将 mnuCut、mnuCopy、mnuPaste 和 mnuSelectAll 菜单项的副本添加到我们的环境菜单中。

然后,代码将一个分隔条添加到菜单中,用来分隔添加的 Edit 菜单项与新的菜单项。最后3行代码使用 cloneMenu 方法来制作添加到 mnuOptions 的字体大小菜单项的副本。用户将可以通过编辑器来更改字体。

使用 cloneMenu 方法是因为 MenuItem 对象不能多次保持使用。通过复制菜单项,可以制作它的副本。虽然它是一个副本,但它仍是一个独立的对象。因此,当启用或禁用最初的菜单项时,也必须制作同样的副本。使用下面的代码更新 InitEdit、InitOptions、InitContext 及 FontSizeChange 事件处理程序的代码:



```
private void InitEdit(Object source, Event e)
{
    //根据选中的文本判断剪切、拷贝和删除菜单项的状态
    if (txtDocument.getSelectedText().length() == 0)
    {
        //在 Edit 菜单中禁用剪切、拷贝和删除菜单项
```

```
mnuCut.setEnabled(false);
mnuCopy.setEnabled(false);
mnuDelete.setEnabled(false);
// 在环境菜单中禁用剪切和拷贝菜单项
cmnCut.setEnabled(false);
cmnCopy.setEnabled(false);
}
else
{
    //在 Edit 菜单中启用剪切、拷贝和删除菜单项
    mnuCut.setEnabled(true);
    mnuCopy.setEnabled(true);
    mnuDelete.setEnabled(true);
    //在环境菜单中启用剪切和拷贝菜单项
    cmnCut.setEnabled(true);
    cmnCopy.setEnabled(true);
}
//根据剪贴板中文本的有效性设置粘贴菜单项命令
try
{
    //在 Edit 菜单中启用剪贴菜单项
    mnuPaste.setEnabled(Clipboard.getDataObject()
        .getDataPresent(String.class));
    //在环境菜单中启用剪贴菜单项
    cmnPaste.setEnabled(Clipboard.getDataObject()
        .getDataPresent(String.class));
}
catch (Exception ex)
{
    //在 Edit 菜单中禁用剪贴菜单项
    mnuPaste.setEnabled(false);
    //在环境菜单中禁用剪贴菜单项
    cmnPaste.setEnabled(false);
}

private void InitOptions(Object source, Event e)
{
    //在选项菜单中更新文字回行菜单项
    mnuWordWrap.setChecked(m_bWordWrap);
}
```

```

//清除选项菜单中所有复选标记
mnuSmallFont.setChecked(false);
mnuMediumFont.setChecked(false);
mnuLargeFont.setChecked(false);
//清除环境菜单中所有复选标记
cmnSmallFont.setChecked(false);
cmnMediumFont.setChecked(false);
cmnLargeFont.setChecked(false);

//检查确切的字体大小对象
switch(m_iCurrentFont)
{
case SMALLFONT:
    mnuSmallFont.setChecked(true);
    cmnSmallFont.setChecked(true);
    break;
case MEDIUMFONT:
    mnuMediumFont.setChecked(true);
    cmnMediumFont.setChecked(true);
    break;
default:
    mnuLargeFont.setChecked(true);
    cmnLargeFont.setChecked(true);
}

private void InitContext(Object source, Event e)
{
    InitEdit(source, e);
    InitOptions(source, e);
}

private void FontSizeChange(Object source, Event e)
{
    //检索菜单项对象
    MenuItem checkingMenu = (MenuItem)source;

    if (mnuSmallFont == checkingMenu
        || cmnSmallFont == checkingMenu)
    {
        //设置字体大小为 11
        txtDocument.setFont(new Font(txtDocument.getFont(),
                                     11.0f,
                                     FontSize.CHARACTERHEIGHT));

        m_iCurrentFont = SMALLFONT;
    }
}

```



```

else if (mnuMediumFont == checkingMenu
        || cnnMediumFont == checkingMenu)
{
    //设置字体大小为 16
    txtDocument.setFont(new Font(txtDocument.getFont(),
                                16.0f,
                                FontSize.CHARACTERHEIGHT));
    m_iCurrentFont = MEDIUMFONT;
}
else
{
    //设置字体大小为 24
    txtDocument.setFont(new Font(txtDocument.getFont(),
                                21.0f,
                                FontSize.CHARACTERHEIGHT));
    m_iCurrentFont = LARGEFONT;
}
}

```

ContextMenu 控件提供所有将菜单项添加、删除、插入到环境菜单的方法。无论是使用代码,还是使用菜单编辑器,您都会发现这个过程非常简单灵活。如果想查看该环境菜单的实际效果,那么可以生成编译并运行该工程。当 WFC Jot 打开时,在 Edit 控件内右击,将会看到组合了 Edit 菜单命令及 Options 菜单字体大小菜单项的环境菜单。请注意,当与 Edit 菜单共享的菜单项同时作为应用程序主菜单的菜单项时,它们将被正确地禁用。图 8.11 显示了环境菜单的实际效果。



图 8.11 运行时的 WFC Jot 环境菜单

添加应用程序的 About 框

定义完环境菜单,再来添加一个 About 框。因为我们的应用程序非常简单,所以这个 About 框也很简单。该 About 框只是用来显示应用程序的名称、版本号,以及版权声明等。图 8.12 显示了它在运行时的效果。



图 8.12 运行时的 AboutBox.java

要创建该框,可以将一个新窗体添加到工程中,并将其命名为 AboutBox.java。表 8.7 列出了 About 框窗体的控件及其属性设置。

表 8.7 AboutBox.java 组件及属性设置

组件类型	属性	设置
Form	size.x	411
	size.y	196
	text	AboutWFCJot
	maxButton	false
	minButton	false
	borderStyle	FixedDialog
	startPosition	CenterScreen
Label	name	lblProduct
	foreColor	ActiveCaption
	size.x	365
	size.y	50
	font.size	32char
	font.weight	Bold
	text	WFCJotforWindows
Label	name	lblVersion
	size.x	260
	size.y	20

续表

组件类型	属性	设置
Label	font.size	20char
	text	Version1.0
	name	lblCopyright
	size.x	265
	size.y	20
	font.size	14
Button	text	Copyright(c)1998, MyVirtualCompany
	name	btnOK
	text	OK

图 8.13 显示了 About 框在设计时的外观。



图 8.13 设计时的 AboutBox.java

将下面的代码添加到 MainForm.java 中,来处理 mnuAbout 的单击事件:



```
private void mnuAbout_Click(Object source, Event e)
{
    AboutBox ab = new AboutBox();
    ab.ShowDialog();
}
```

需要自己创建的 AboutBox.java 的代码由 btnOK 控件的 click 事件的处理程序与事件处理程序中的 dispose 调用共同组成。

这个 About 框设计得很简单,因为这里的主要目的是讨论窗口附件。如果需要,可以增强它。为了进一步增强我们的应用程序,需要完成最后的任务。我们必须在用户使用控件框或单击窗口右上角的 Close 按钮关闭应用程序时,检查是否需要保存文档。因此,我们需要将事件处理程序添加到 MainForm.java 中,来处理 closing 事件。使用默认的名称 MainForm_closing 命名该事件处理程序,

然后添加下面的代码：



```
//当单击 Close 按钮关闭应用程序时,检查文档是否为旧文档
private void MainForm_closing(Object source, CancelEvent e)
{
    // 检查旧文档标志是否为 false
    if (HandleDirt(source, e) == false)
    {
        //停止关闭应用程序...
        e.cancel = true;
    }
}
```

该代码类似于 mmuExit_click 事件的代码,除了它调用 HandleDirt 并检查它来返回值之外。如果返回值为 false,那么代码将 CancelEvent 对象的 Cancel 变量设置为 true。将该变量设置为 true,将会取消关闭应用程序。简单地添加这些内容,就可以避免用户无意丢失数据的尴尬局面。

8.4 创建工具栏

工具栏是另一种方便的窗口附件,它可以使您的应用程序使用更方便、更具吸引力。Visual J++ 的 ToolBar 控件提供了灵活强大的方法,来将工具栏支持添加到应用程序中。使用 ToolBox 控件,可以添加具有文本或图像的按钮。

按钮不是唯一可添加的工具栏工程类型。还可以定义具有相关下拉菜单的按钮。当用户想通过简单地访问来显示工具栏菜单的子集时,这种功能非常有用。另外,工具栏还为菜单项提供了内置的工具提示支持,以使您的用户可以更加容易地识别按钮及其功能。ToolBar 控件还支持像传统工具栏那样以标准三维模式,或类似于 Visual J++ 工具栏按钮那样以平面模式来显示按钮。正如您所看到的,ToolBar 控件具有无比的灵活性来适应几乎所有的需要。

在本节中,我们将调用该应用程序中定义的菜单命令的工具栏添加到 WFC Jot 应用程序中。还将添加显示 Options 菜单项的工具栏按钮。完成本节后,我们的工程将更加接近于实际的应用程序。图 8.14 显示了添加 ToolBar 控件后的 WFC Jot 应用程序。



图 8.14 具有工具栏的 WFC Jot 应用程序

8.4.1 在新工程中打开 WFC Jot 应用程序

如果使用本书附带光盘中的代码来继续该源代码,那么您可以打开 Samples \chap08 \WFCJot4 目录下的 WFCJot 工程。否则,复制 Samples \chap08 \WFCJot3 目录下的文件,然后将它们放到名为 WFCJot4 的新目录下(这是为了确保先前的工程不会受到添加内容的影响)。

8.4.2 添加 ImageList 控件及其图像

打开 MainForm.java,以便显示 Forms Designer。在该步骤中,将添加存储工具栏按钮图像的 ImageList 控件。将 ImageList 控件添加到窗体中,并将其命名为 imglButton。该控件不提供任何用户界面,所以可以将它添加到窗体的任意位置。要将图像分配给该控件中存储的列表,可以选定 image 属性,然后单击该属性中的 Ellipsis 按钮。这时,将显示如图 8.15 所示的对话框。在开发人员需要指定工程集合位置的其他 Visual J++ 控件时使用该对话框。要将图像添加到列表中,可以单击 Add 按钮。这时将显示一个标准的 File 的 Open 对话框,以便您可以定位到图像文件,然后将它添加到列表中。

我们将要添加到 ImageList 的按钮图标位于 Visual J++ 安装目录——Microsoft Visual Studio 下的 Common \Graphics \Bitmaps \Tlbr...w95 目录下。本书附带光盘的 Samples \chap08 \WFCJot4 目录下也有这些位图文件。使用表 8.8 将位图文件添加

到 `ImageList` 控件中。这个顺序很重要,因为此时也将使用图像的索引来确定工具栏上被单击的按钮。当然也可以使用个别工具栏对象来确定被单击的按钮,但是使用图像索引更易于操作,因为它是一个整数,并且可以在切换语句中使用。



图 8.15 `ImageList` 控件的图像阵列编辑器

表 8.8 `ImageList` 图像

位图文件	索引
<code>new.bmp</code>	0
<code>open.bmp</code>	1
<code>save.bmp</code>	2
<code>cut.bmp</code>	3
<code>copy.bmp</code>	4
<code>paste.bmp</code>	5
<code>delete.bmp</code>	6
<code>find.bmp</code>	7
<code>prop.bmp</code>	8

我们的工具栏位图存储在将这些位图添加到 `ImageList` 中时自动创建的 `MainForm.resources` 文件中。这意味着,我们不必将位图存储在应用程序中,而只需存储在一个资源文件中。

8.4.3 添加 `ToolBar` 控件及其按钮

为 `ImageList` 配置完工具栏按钮图像,下面要添加到窗体中的内容是 `ToolBar` 控件及其按钮。将 `ToolBar` 控件添加到 `MainForm.java` 中,然后将其命名为 `tlb-`

Main。可以将它放在窗体的任意位置。要添加按钮,可以单击 `buttons` 属性值区旁的 `Ellipsis` 按钮。这将打开按钮阵列编辑器,它几乎与图 8.15 所示的 `Image Editor` 对话框相同。单击 `Add` 按钮,将按钮添加到按钮阵列中。

我们总共将 11 个按钮添加到工具栏中。还将根据与每个按钮相关的数字来引用按钮。添加完所有的按钮后,单击 `OK` 按钮。`Properties` 窗口现在将在添加的每个按钮的 `buttons` 属性下显示一系列节点。`all` 节点表示所有按钮的全局属性。因为我们的应用程序除了工具提示之外,不打算显示其他任何文本,所以可以清除每个按钮 `text` 属性中的文本。要使 `imglButton` 的图像对于每个按钮的 `imageIndex` 属性可用,可以将工具栏的 `imageList` 属性设置为 `imglButton`。可以从为属性值提供的下拉列表选择。要配置其余的工具栏按钮属性,可以使用表 8.9 显示的内容。

表 8.9 工具栏按钮属性

按钮	toolTipText	样式	dropDownMenu	imageIndex
0	New Document	Pushbutton	None	0
1	Open Document	Pushbutton	None	1
2	Save Document	Pushbutton	None	2
3	"	Separator	None	-1
4	Cut	Pushbutton	None	3
5	Copy	Pushbutton	None	4
6	Paste	Pushbutton	None	5
7	Delete	Pushbutton	None	6
8	"	Separator	None	-1
9	Find And Replace	Pushbutton	None	7
10	Options	Dropdownbutton	menuOptions	8

设置完按钮,现在需要设置一些属性来达到需要的工具栏效果及外观。使用表 8.10 来设置这些属性。

表 8.10 工具栏属性设置

属性名称	值
<code>buttonSize.x</code>	16
<code>buttonSize.y</code>	16
<code>DropDownArrows</code>	true
<code>ShowToolTips</code>	true
<code>Appearance</code>	Flat

`buttonSize` 属性定义所有按钮的大小。标准的 Windows 应用程序通常采用 16

像素×16像素的工具栏按钮。可以为具有指定的 `dropDownMenu` 类型的按钮设置工具栏的 `dropDownArrows` 属性。如果将 `dropDownArrows` 属性设置为 `true`, 那么 `dropDownMenu` 按钮将在按钮旁显示一个箭头, 以便用户可以选择它来显示下拉菜单。

工具栏带有内置的工具提示支持; 将 `showToolTips` 设置为 `true` 可以启用工具栏的这个支持。默认情况下, 工具栏将每个按钮显示为一个独立的三维按钮。具有三维按钮的工具栏在样式上更为传统。Office 97 和 Internet Explorer 4.0 以上的版本是平面形式的按钮, 只有当用户将鼠标指针放在按钮上时, 它才会变成三维效果。将 `appearance` 属性设置为 `Flat` 对于工具栏来说非常有特点。

正如您所看到的, 创建工具栏非常简单。在下一小节中, 我们将添加一些事件处理程序, 以使工具栏能够正常工作。

8.4.4 处理工具栏的事件

我们的工具栏接近完成了。现在只需添加一些事件处理程序, 第一个将处理 `buttonClick`, 当用户单击任意的工具栏按钮时, 都会触发该事件。通过该事件的处理程序, 可以确定被单击的按钮, 因为我们的工具栏是以菜单事件处理程序调用的菜单为基础的。为 `sbMain` 的 `buttonClick` 事件创建一个事件处理程序, 并将其命名为 `TbuttonClicked`。将下面的代码添加到处理程序中:



```
private void TbuttonClicked(Object source, ToolBarButtonClickEvent e)
{
    switch(e.button.getImageIndex())
    {
        case 0: //新建文档
            mnuNew_click(source, e);
            break;
        case 1: //打开文档
            mnuOpen_click(source, e);
            break;
        case 2: //保存文档
            mnuSave_click(source, e);
            break;
        case 3: //剪切
            mnuCut_click(source, e);
            break;
        case 4: //拷贝
```

```

        mnuCopy_click(source,e);
        break;
    case 5: //粘贴
        mnuPaste_click(source,e);
        break;
    case 6: //删除
        mnuDelete_click(source,e);
        break;
    case 7: //查找
        this.mnuFindReplace_click(source,e);
        break;
    }
}

```

回顾一下,在将按钮添加到工具栏并设置每个按钮的属性时,我们从来就没有为它们指定过名称。在添加按钮时,Visual J++ 为每一个按钮提供了一个索引。不幸的是,该索引在作为参数传递的 `ToolBarButtonClickEvent` 对象 `e` 的参数中是不可用的。但是,我们仍可以确定触发事件的按钮。工具栏上的每一个按钮,除了分隔条按钮,都是从添加的 `ImageList` 控件 `imglButton` 处获得一个 `imageIndex`。因为每个图像都只能使用一次,所以该索引值对于每个按钮是唯一的。

在上面的代码中,我们使用了 `ToolBarButtonClickEvent` 对象中 `button` 对象的 `getImageIndex` 方法。`getImageIndex` 方法返回一个整数来表示 `ImageList` 控件中图像的索引。然后切换语句根据这个值来确定用户单击的按钮。根据用户单击的按钮,与该按钮相关的菜单项的事件处理程序就会被调用。该代码不能处理的唯一一个按钮是最后一个按钮——`DropDownMenu` 型按钮,与该按钮相关的菜单响应 `buttonClick` 事件来进行操作。因为我们使用一个事件处理程序来管理所有的按钮单击事件,所以将所有的按钮放在一个位置是很方便的。事实上,利用事件处理程序能够确定被点击按钮的功能,我们可以在运行时添加按钮,并且可以很容易地处理它们。

我们还需要最后一些代码来完成工具栏,并改进它的功能。回顾前面创建菜单事件处理程序的章节,我们知道 `mnuCut`、`mnuCopy`、`mnuPaste` 和 `mnuDelete` 菜单项都是根据 `Edit` 控件中选定文本或【剪贴板】中可用于粘贴的 `String` 数据来启用或禁用。启用和禁用都发生在 `mnuEdit` 菜单的 `popup` 事件中,所以菜单会正确地显示。工具栏应该执行相同的方法,并且相应地禁用和启用 `Cut`、`Copy`、`Paste` 和 `Delete` 按钮。但是,我们使用什么事件来更新按钮呢?

`com.ms.wfc.app.Application` 类为与应用程序的操作进行交互提供了一些静

态方法。它还提供了一些关于应用程序范围的事件,其中包括 idle 事件。当应用程序不处理任何信息或用户操作而发生的事件时,应用程序将获得 idle 事件。通常情况下,该事件执行一些清理事务,例如清除临时文件等。对于处理更新工具栏,这是一个非常完美的事件。在下一节中,我们还将使用它来更新状态栏。因为 Application 类不是图形化的,所以不能使用 Properties 窗口来创建事件处理程序。我们必须添加代码。将下面的代码添加到 MainForm.java 构造函数的结尾处。该代码将为 idle 事件创建一个事件处理程序,来指向我们添加的 Update-Toolbar 事件处理程序。



```
//添加处理应用程序 idle 事件的事件句柄来更新工具栏
Application.addOnIdle(
    new EventHandler(this, "UpdateToolbar"));
```

下面需要添加当触发 idle 事件时调用的事件处理程序。将下面的方法添加到 MainForm.java 中:



```
//更新工具栏按钮的 idle 事件句柄
private void UpdateToolbar(Object source, Event e)
{
    //如果文本被选中,启用工具栏的 Cut, Copy 和 Delete 按钮
    if (txtDocument.getSelectedText().length() == 0 &&
        tlbMain.getButton(4).getEnabled() == true)
    {
        tlbMain.getButton(4).setEnabled(false);
        tlbMain.getButton(5).setEnabled(false);
        tlbMain.getButton(7).setEnabled(false);
    }
    else if (txtDocument.getSelectedText().length() != 0 &&
        tlbMain.getButton(4).getEnabled() == false)
    {
        tlbMain.getButton(4).setEnabled(true);
        tlbMain.getButton(5).setEnabled(true);
        tlbMain.getButton(7).setEnabled(true);
    }

    //在剪贴板中文本有效性的基础上设置 Paste 按钮
    try
    {
        boolean dataAvailable = Clipboard.getDataObject()
```

```

        .getDataPresent(String.class);
        if (tlbMain.getButton(6).getEnabled() != dataAvailable)
            tlbMain.getButton(6).setEnabled(dataAvailable);
    }
    catch (Exception ex)
    {
        // 在工具栏上禁用 Paste 按钮
        tlbMain.getButton(6).setEnabled(false);
    }
}

```

该代码类似于前面添加的用来初始化 Edit 菜单的 `initEdit` 方法。两者的区别在于该代码设置工具栏按钮的启用状态,而不是菜单项。该代码可能会造成工具栏随着每次启用或禁用调用而闪烁,因为当应用程序空闲时会触发该代码。要避免这种闪烁,代码不仅要查看 `txtDocument` 是否包含有选定的文本或者【剪贴板】有要粘贴的文本,而且还要检查所涉及的工具栏按钮是否已经被设置适当的值。如果是,那么代码将略过启用或禁用按钮的语句。

如果用户在应用程序中操作迅速而来不及触发 `idle` 事件怎么办?不用担心。该事件会经常触发,因此可以避免不断地更新工具栏按钮。

现在已经处理了工具栏按钮的启用和禁用问题,也就完成了将工具栏添加到 WFC Jot 中的任务。编译应用程序并运行它。应用程序加载后,注意我们添加的新工具栏。在 Save Document 按钮与 Cut 按钮之间,以及 Delete 按钮与 Find And Replace 按钮之间都有一个分隔条。试着像使用菜单那样使用工具栏来打开和保存文件,并操作文档。单击 Options 下拉箭头。控件将显示 Options 菜单项,并且在适当的字体大小菜单项旁设置有复选标记。

8.5 创建状态栏

状态栏与前面讨论的大多数窗口附件不同,它可以作为应用程序的一个集成部分。如果设计的是图形程序,那么用户需要了解鼠标当前的 `x,y` 坐标。编译数据的程序需要显示应用程序的状态信息。几乎各种应用程序都需要用状态栏来显示重要的应用程序信息。

在这一节中,我们将把状态栏添加到 WFC Jot 中。状态栏将显示日期和时间信息,以及报告打开和保存文件的状态区。图 8.16 显示了添加 `StatusBar` 控件后的 WFC Jot 应用程序。我们的状态栏的用途有限,但本节将提供足够的信息,以便在应用程序中能够有效地使用状态栏。



图 8.16 添加 StatusBar 控件后的 WFC Jot 应用程序

8.5.1 在新工程中打开 WFC Jot 应用程序

如果使用本书附带光盘上的代码来继续该源代码,那么可以打开 Samples \ chap08 \ WFCJot5 目录下的 WFCJot 工程。否则,复制 Samples \ chap08 \ WFCJot4 目录下的文件,然后将它们放在名为 WFCJot5 的新目录下,以便我们添加的内容不会影响到先前的工程。

8.5.2 添加 StatusBar 控件并创建其窗格

打开工程,选定 MainForm.java 文件,然后在 Forms Designer 中将其打开。从工具箱中选定 StatusBar 控件,然后将其添加到窗体中。将该控件命名为“sbar-Main”。像前面章节添加的 ToolBar 控件一样,StatusBar 控件会自动停放在窗体的底部。我们将在 StatusBar 控件内创建 3 个窗格。第 1 个窗格占用状态栏的大部分空间,并显示应用程序的状态信息。第 2 个窗格显示当前日期;最后 1 个窗格显示当前时间。添加这些窗格就像将按钮添加到 ToolBar 控件中一样简单。

要添加这 3 个窗格,可以单击 panels 属性,然后单击该属性值区的 Ellipsis 按钮。这时,将显示一个对话框,它类似于前面在将图像添加到 ImageList 控件,以及将按钮添加到 ToolBar 控件中时使用的对话框。单击 3 次 Add 按钮,创建 3 个窗格,然后单击 OK 按钮,退出该对话框。3 个窗格将列举在 panels 属性下。可以使用表 8.11 为每个窗格设置属性。

表 8.11 StatusBar 窗格属性

窗格序号	对齐方式	autoSize	文本
0	Left	Spring	Ready
1	Center	Contents	" "
2	Center	Contents	" "

另外,将 sbarMain 的 showPanels 属性设置为 true,以便 StatusBar 控件显示新的窗格。如果保持 showPanels 设置为 false,那么 StatusBar 控件只显示它的 text 属性(不是每个窗格的 text 属性)的内容。

值得讨论的一个属性是 autoSize。每一个窗格都具有 Contents 的 autoSize 属性设置,这意味着窗格将与窗格中的文本内容等宽。第一个窗格的 autoSize 属性设置为 Spring,将使该窗格占用尽可能大的空间。因为第一个窗格可以显示大量文本,所以要确保将窗格的 autoSize 属性设置为 Spring。

现在状态栏已经完成。最后一步将添加少量代码来支持状态栏。

8.5.3 添加支持状态栏的代码

我们将使用状态栏来显示日期和时间,并且还要显示与在应用程序中打开和保存文档相关的信息。让我们先来添加代码。将下面的代码添加到方法 try 代码块下面的 mmuOpen_click 事件处理程序中:



```
//在状态栏中通知用户所打开的文件名
sbarMain.getStatusBarPanel(0).setText("Opened file" + m_sFilename);
```

另外,将下列代码添加到 SaveFile 方法的 else 块的后面,以及 SaveFileAs 方法 ChangeDialogCaption 调用的后面:



```
sbarMain.getStatusBarPanel(0).setText("Saved file" + m_sFilename);
```

在这两个代码段中,我们调用 getStatusBarPanel 来检索代表第一个窗格的 StatusBarPanel 对象。使用该窗格的引用,代码将调用 setText 方法来设置文本。在这两种情况下,将通知用户在 m_sFilename 中指定的文件被打开或保存。如果您想将这种类型的支持添加到应用程序中,那么可以使用这些代码作为一个实例。

最后需要添加日期和时间窗格的支持代码。因为我们将经常更新这些窗格,以确保日期和时间保持是最新的,所以将添加显示 UpdateToolbar 事件处理程

序中的日期和时间的代码。对于这种类型的更新,Idle 事件是非常完美的。将下面的代码添加到 UpdateToolBar 事件处理程序的后面:



```
//更新状态栏中的时间
sbarMain.getStatusBarPanel(2).setText(new Time().formatLongTime());
//在状态栏中第二个窗格中设置日期显示
sbarMain.getStatusBarPanel(1).setText(new Time().formatShortDate());
```

我们只需要两行代码来更新状态栏最后两个窗格中的文本。第1行代码像前面添加的代码那样,使用 `getStatusBarPanel` 和 `setText` 方法。为了计算时间,代码创建了一个 `com.ms.wfc.app.Time` 类的实例。代码调用该实例将当前时间以 `hh:mm am/pm` 格式作为 `String` 返回的 `formatLongTime` 方法。第2行代码除了调用 `Time` 类的 `formatShortDate` 方法以短日期格式 `mm/dd/yy` 返回 `String` 之外,几乎与前面一行完全相同。

现在我们已经完成了 `StatusBar` 控件及其窗格。编译工程并运行程序,您会注意到,状态栏被添加到窗体中。当您打开一个文件,状态栏的第一个窗格将显示文件的状态——打开、保存或更新。如果保持应用程序长时间运行,那么您会注意到状态栏中的日期和时间在不断地更新。



更上一层楼

在本章中,我们学习了创建和显示窗体、创建屏幕保护程序等有关窗体的一些内容,还学习了菜单、工具栏和状态栏等窗口附件的创建和使用。

关于窗体,您还可以进一步学习创建不同类型的窗体,例如给屏幕保护程序增加设置选项对话框,创建子类化窗体,制作工具窗口等。我们前面做的窗体都比较简单,可以尝试 WFC 提供的丰富控件使自己的窗体更加漂亮、实用。有关窗体的更多的属性、方法和事件也是需要进一步学习的内容。

关于窗口附件,我们前面添加了许多功能来在原有的程序基础上不断添加新功能。可以考虑以下问题来进一步增强 WFC Jot 应用程序的功能:

- ◆ 通过建立 MDI 应用程序,在 File 菜单中添加最近使用过的文件列表,这样您就可以快速访问以前打开过的文件。
- ◆ 添加一个显示字体对话框(`FontDialog`)的选项,以使用户可以选择字体设置,不仅可以选择字体在大小,还可以选择字体的类型。
- ◆ 现在的 WFC Jot 应用程序只能显示文本文件。要增强支持,可以使用

RichEdit 控件来显示 RTF 文档及文本文件。您可能还希望添加更多字体和颜色更改的支持,以便创建具有不同字体和不同颜色的文本的文档。

- ◆ WFC Jot 应用程序支持 Cut、Copy、Paste 以及 Delete。如何实现允许用户恢复刚刚进行更改的 Undo 功能呢?您可以使用 `txtDocument` 的 `change` 事件和应用程序的 `idle` 事件来捕获 `txtDocument` 的更改,以便可以恢复这些更改,实现 Undo 功能。



第9章 Java 应用程序编程进阶

知识要点:

- ◆ 综合使用 Windows 基本控件
- ◆ 如何将帮助添加到应用程序中
- ◆ 如何在应用程序中支持拖放
- ◆ ActiveX 有关概念
- ◆ 如何在 WFC 应用程序中使用 ActiveX 控件
- ◆ Java 应用程序与 Java 小程序的混合

在第8章中,我们讨论了编写应用程序的几个方面,包括窗体、菜单、工具栏、状态栏等。对于习惯 Windows 图形用户界面的用户来讲,应用程序还应包括详细的帮助、支持简单的拖放操作。本章首先用一个综合例子说明如何使用控件,以此为基础讲述如何增加帮助,接着讨论在应用程序中使用拖放的例子,然后讨论有关 ActiveX 的技术。COM 作为一种编程模型已广泛应用于 Windows 平台中,ActiveX 是建立在 COM 基础之上,融入 Internet 特性的一种技术。怎样把 ActiveX 控件导入到 WFC 应用程序中,是我们讨论的重点。最后,我们讨论 Java 应用程序与 Java 小程序的混合,即写一个 Java 执行码程序,即可作为独立的 Java 应用程序运行,也可作为 Java 小程序在浏览器中运行。

9.1 使用控件

随着 Windows 在世界范围内成为大多数计算机的基本操作系统,人们对功能更加强大的应用程序的需求也与日俱增。开发人员需要一种方法将新的技术和用户界面集成到他们的应用程序中。Windows 已经提供了一些默认的用户界面元素,例如按钮、列表框、复选框等,但是开发人员需要新的能够处理应用程序界面需求的可重用元素。他们还需要能够在其他应用程序中及通过各种编程语言访问的可重用的用户界面元素。作为这种需求的回应,Microsoft 创建了我们所说的“控件”。

9.1.1 什么是控件

控件是封装了一些功能,以及应用程序用来与用户进行交互或执行任务的组件或对象。控件的优点之一是,使用它们可以创建显示在应用程序中的新型用户界面。无论是试图定义新的应用程序外观,还是提供一种更为简便的访问数据的方法,自定义控件都是比较合适的工具。例如,我们为飞机制造商开发的一个应用程序要求在一个网格中显示每个记录的两行数据。我们能够找到提供这种所需网格的控件。我们所要做的就是将它放到我们的应用程序中,添加一些支持代码,然后,我们的应用程序就可以适应用户的要求。

控件的另一个优点是,可以将它们用于非用户界面任务。例如,Timer 控件不提供用户界面,而是提供一些其他功能。将一组方法封装到一个可重用的拖放组件中的能力可以为开发人员节约大量时间。

Visual J++ 6.0 在 Windows 基类(Windows Foundation Classes, WFC)应用程序中支持两种不同类型的控件。第一种类型是 WFC 控件,我们在前面章节的示例中已经使用过它们。这些控件被设计用来访问 Java 以优化 WFC 应用程序的特性和功能。第二种,是在 Visual J++ 中使用的、更为常见的类型——ActiveX 控件。这种控件是 Visual Basic、Visual C++ ,以及其他一些应用程序(例如,Microsoft Office 中包含的应用程序)的支柱。ActiveX 控件提供大量功能,并且可以快速方便地导入到 WFC 应用程序中。许多第三方销售商还提供了一些可以在您的应用程序中使用的独特的 ActiveX 控件。

9.1.2 综合使用 Windows 基本控件

正如前面所提到的,Windows 为基本用户界面提供了一些基本控件,其中包括 Button、Label、CheckBox、ListBox、PictureBox、RadioButton、ComboBox,以及 GroupBox 控件等。您可以使用 Win32 API 来访问它们,但是 WFC 通过将它们封装到控件类中消除了创建这些控件的复杂性。现在您可以使用标准的面向对象的代码来访问这些控件。

为了演示如何使用基本控件,以及如何组合它们来创建有效的用户界面,我们将查看一个为选拔冰球少年联赛运动员编写的报告管理程序。该应用程序被设计为存储选拔人员掌握的运动员的基本信息,例如每名运动员的技术和缺点。图 9.1 和图 9.2 显示了该应用程序在运行时的界面。

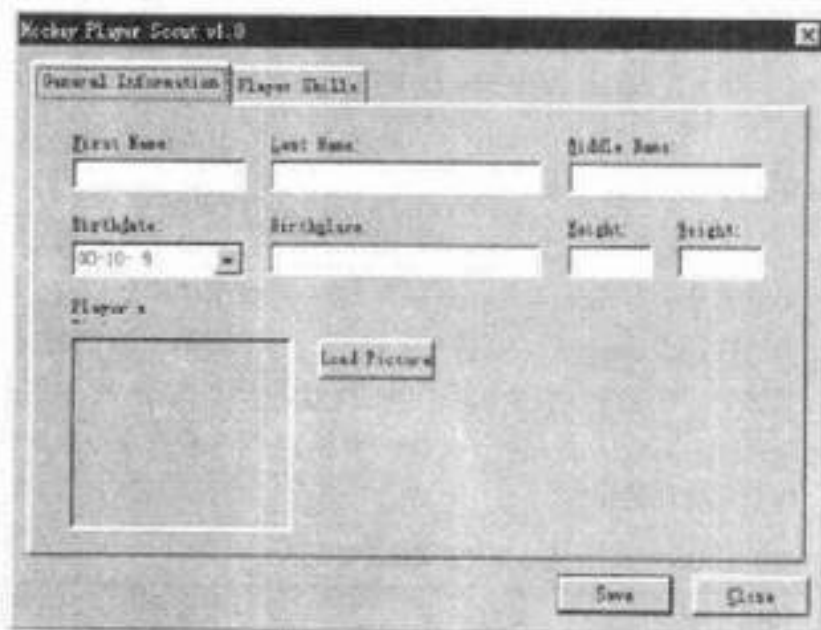


图 9.1 Hockey Player Scout 运行时的第一个选项卡

该示例允许我们使用几乎所有重要的基本控件,以及其他一些有用的 WFC 控件。虽然这个应用程序比较简单,它仍提供了足够的功能,并且非常有用。在本章稍后的示例中,我们将利用该应用程序通过显示运动员资料的页面程序生成的数据。要简化数据的读写过程,应用程序将以 .ini 文件格式创建文件。在第 11 章“数据库编程”中,我们将转换该应用程序,以使它不再在文件中存储信息,而是使用数据库来存储信息。

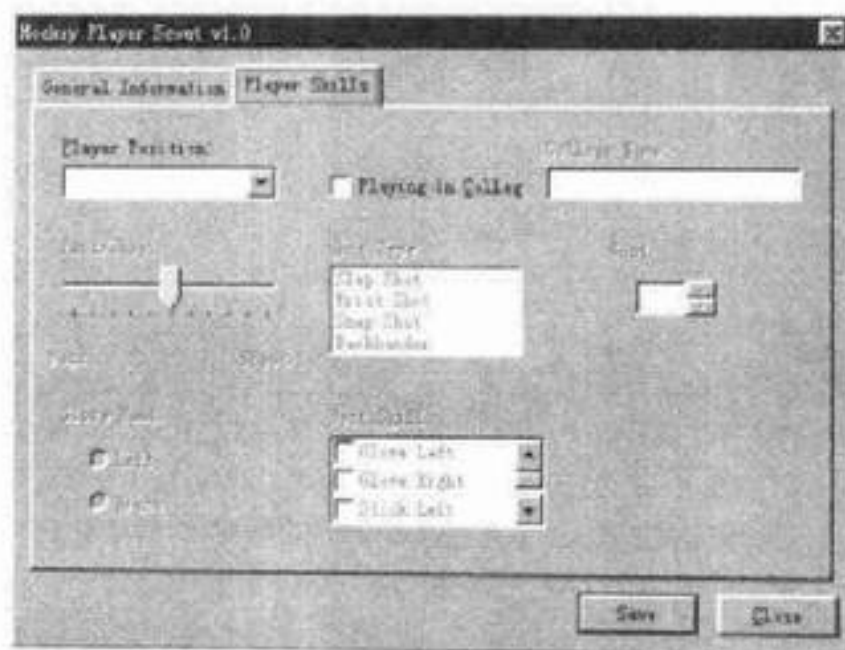


图 9.2 Hockey Player Scout 运行时的第二个选项卡

创建工程并添加窗体

要开始构建选拔应用程序,可以使用 Empty Project 模板创建一个新的工程,并将其命名为“HockeyPlayerScout”。将一个新窗体添加到该工程中,并将其命名为“HockeyScout.java”。从图 9.1 和图 9.2 可以看出,我们将把一些控件添加到窗体中。在这一步剩余的工作中,我们将添加 Button 控件、SaveFileDialog 控件,以及 TabControl。使用表 9.1 中的设置,将这些控件添加到窗体中,然后指定窗体及其控件的属性。

表 9.1 HockeyScout.java 组件及属性设置

组件类型	属性	设置
Form	size.x	527
	size.y	361
	StartPosition	Center Screen
	Text	Hockey Player Scout v1.0
	AcceptButton	btnSave
	CancelButton	btnClose
	borderStyle	Fixed Dialog
	minimizeBox	false
	maximizeBox	false

续表

组件类型	属性	设置
SaveFileDialog	Name	sfdSavePlayer
	DefaultExt	.player
	Filter	Player Record (* .player) * .player All Files(* .*) * . *
TabControl	Title	Select name and location for player...
	name	abMain
	tabIndex	0
Button	name	btnSave
	text	Save
	tabIndex	1
Button	name	btnClose
	Text	&Close
	tabIndex	2



注意 在可以设置窗体中的 `acceptButton` 和 `cancelButton` 属性之前,需要在两个 `Button` 控件中设置 `name` 属性。

图 9.3 显示了添加完控件后的窗体外观。

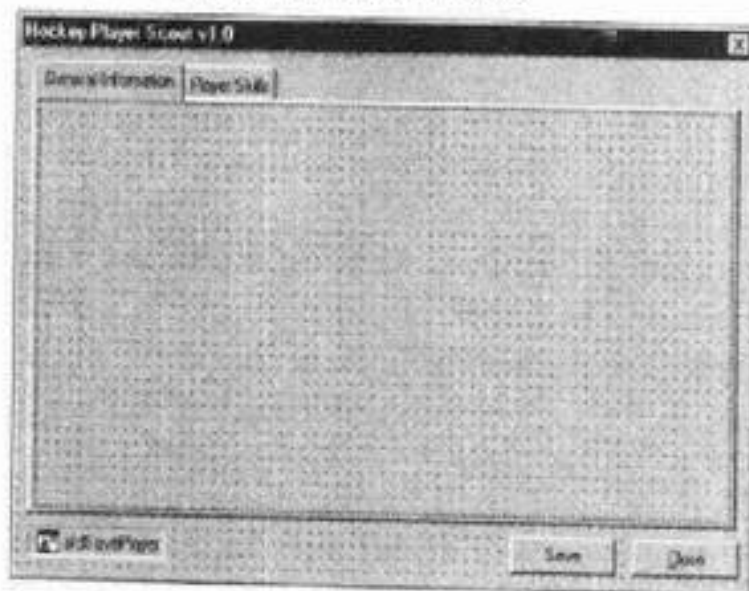


图 9.3 设计时添加基本控件后的 HockeyScout.java

`TabControl` 使您可以显示控件的多个页面。当需要显示一个窗体的多个页面时,它是非常有用的。要创建如图 9.3 所示的选项卡,可以在 `Properties` 窗口

中单击 TabControl, 然后选定 tabs 属性。单击属性值区右边的 Ellipsis 按钮, 将显示 Tab Array Editor 对话框。单击两次 Add 按钮, 将两个选项卡添加到控件中。单击 OK, 返回到 Properties 窗口。

现在 tab 属性旁将有一个加号 (+) 框。单击该框, 显示选项卡的属性。您将注意到每个选项卡都有诸多属性。对于每个选项卡, 可以指定诸如字体、用选项卡显示的图像, 以及工具提示等信息。这里, 我们只指定在每个选项卡上显示的文本, 以及选项卡的名称。使用表 9.2 指定每个选项卡的属性。

表 9.2 HockeyScout.java 选项卡属性设置

Tab 索引	属性	设置
0	Name	tpGeneral
	Text	General Information
1	Name	tpSkills
	text	Player Skills

添加到 TabControl 中的每个选项卡都包含一个与该选项卡相关的 tabPage 对象。TabPage 是一个表示可以将控件添加到相关选项卡位置的容器对象。在大多数情况下, 当与不同的选项卡进行交互时, 都需要响应 tabPage 对象而不是 TabControl 对象的事件。SaveFileDialog 是一个相当于 SaveFileDialog 类的控件。使用控件, 我们就可以使用 Properties 窗口来指定 SaveFileDialog 的配置。接下来, 我们将把控件添加到在这一步中添加的两个选项卡中。

将控件添加到 General Information 选项卡中

首先, 将一些控件添加到 General Information 选项卡中。该选项卡将显示运动员的基本信息, 例如姓名、生日、出生地、身高, 以及体重等。我们还将提供显示指定运动员图像的支持。因为冰球队选拔人员可能会在几百名运动员中进行搜索, 所以他们大多都是使用相关的图像和姓名来标识运动员。我们的应用程序将使相关的图像和姓名作为选拔人员区别运动员的首选方法。使用表 9.3, 将控件添加到该选项卡中, 并指定它们的属性设置。

表 9.3 HockeyScout.java(tpGeneral) 组件及属性设置

组件类型/ID	属性	设置
Label - 0	name	lblFirstName
	text	&First Name:
Edit - 1	name	txtFirstName
	text	""

续表

组件类型/ID	属性	设置
Label - 2	name	lblLastName
	text	&Last Name:
Edit - 3	name	txtLastName
	text	""
Label - 4	name	lblMiddleName
	text	&Middle Name:
Edit	name	txtMiddleName
	text	""
Label - 6	name	lblBDay
	text	Birth&date:
DateTimePicker - 7	name	dtpBDay
	minDate	12/31/1970
	format	Short
Label - 8	name	lblBirthplace
	text	&Birthplace:
Edit - 9	name	txtBirthplace
	text	""
Label - 10	name	lblHeight
	text	&Height:
Edit - 11	name	txtHeight
	text	""
Label - 12	name	lblWeight
	text	&Weight:
Edit - 13	name	txtWeight
	text	""
Label - 14	name	lblPicture
	text	Player's Picture:
PictureBox - 15	name	picPicture
	borderStyle	Fixed3d
Button - 16	name	btnLoadPicture
	text	Load Picture



注意 表中为每个组件指定的 ID 号与该控件的 `tabIndex` 值标识了图 9.4 中的控件。

图 9.4 显示了第一个选项卡在添加完控件并设置完属性后的界面。

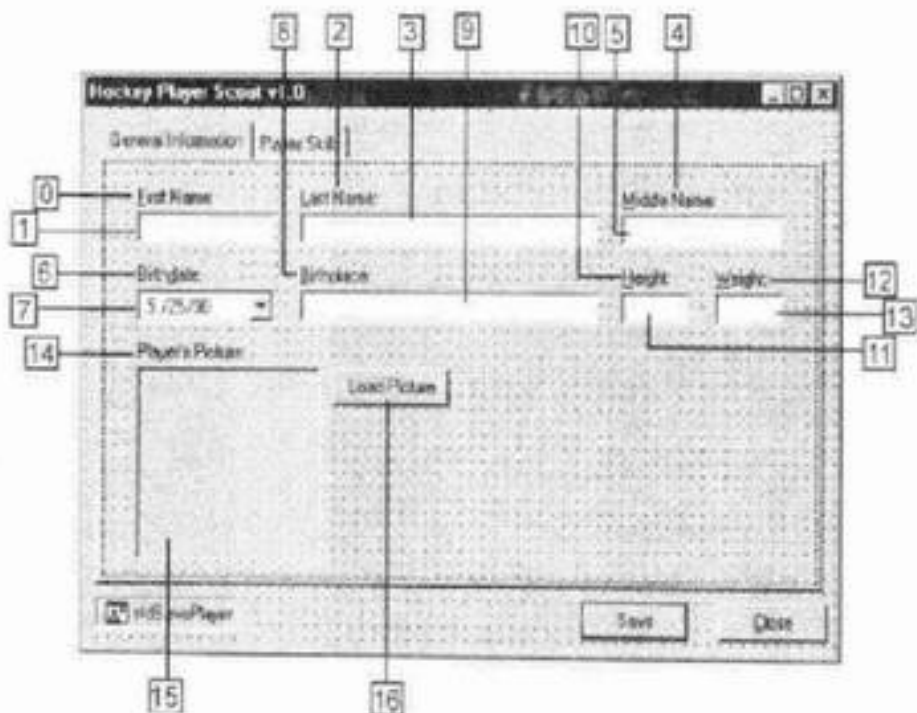


图 9.4 添加控件后的 General Information 选项卡

我们添加的大多数控件都是 Label 和 Edit 的组合,因此它们很容易理解。DateTimePicker 控件允许用户输入短格式日期,或者从单击下拉箭头时显示的日历中选择一个日期。我选择的最早日期为 12/31/70,因为您发现候选运动员的出生日期大都在该日期之后。将 format 属性设置为 Short,以确保日期以短格式存储。在代码中,我们将把该控件初始化为当前日期,以确保我们具有有效的初始日期值。PictureBox 控件使我们可以将运动员的照片加载到记录中。为了加载照片,提供了一个用来显示选择图像的 OpenFileDialog 对话框的按钮。当我们开始添加代码时再详细介绍这些内容。

将控件添加到 Player Skills 选项卡中

在 Player Skills 选项卡中,我们将使用更多的基本控件,例如 CheckBox、ComboBox、ListBox,以及 RadioButton 等。除了基本控件之外,我们还将使用其他一些控件,如 TrackBar、CheckedListBox 及 UpDown 等。使用表 9.4,添加控件并设置它们的属性。

表 9.4 HockeyScout.java(tpSkills)组件及属性设置

组件类型/ID	属性	设置
Label - 0	name	lblPosition
	text	&Player Position:
ComboBox - 1	name	cmbPosition
	style	Dropdownlist
CheckBox - 2	name	chkCollege
	text	Playing in &College?
Label - 3	name	lblCollege
	text	College &Name:
	enabled	false
Edit - 4	name	txtCollege
	text	..
	enabled	false
Label - 5	name	lblPassShoot
	text	Pass/Shoot Ratio:
	enabled	false
TrackBar - 6	name	trbPassShoot
	maximum	10
	minimum	0
	value	5
	enabled	false
Label - 7	name	lblPass
	text	Pass
	enabled	false
Label - 8	name	lblShoot
	text	Shoot
	enabled	false
Label - 9	name	lblShotType
	text	Shot &Type:
	enabled	false
ListBox - 10	name	lstShotType
	enabled	false
Label - 11	name	lblShotAc
	text	Shot Accuracy:
	enabled	false

续表

组件类型	属性	设置
Edit - 12	name	txtShotAc
	text	""
	acceptsReturn	false
	maxLength	2
	readOnly	true
	enabled	false
UpDown - 13	name	udShotAc
	alignment	Right
	value	0
	buddyControl	txtShotAc
	minimum	0
	maximum	10
	enabled	false
Label - 14	name	lblGloveHand
	text	Glove Hand:
	enabled	false
RadioButton - 15	name	optLeft
	text	Le&ft
	enabled	false
RadioButton - 16	name	optRight
	text	&Right
	enabled	false
Label - 17	name	lblSkillLoc
	text	Best S&kill Location:
	enabled	false
CheckedListBox - 18	name	clstSkillLoc
	checkOnClick	true
	threeDCheckBoxes	true
	enabled	false



注意 表中为每个组件指定的 ID 号与该控件的 `tabIndex` 值标识了图 9.5 中的控件。

参照图 9.5 设计窗体。

正如您所看到的,该选项卡具有各种类型的控件。其中我们添加的 3 个控件显示了选项列表。我们可以使用代码来将这些项目添加到列表中,但是 Visual

J++ 还提供了一种非常简单的使用 Properties 窗口来添加项目的方法。要将列表项目添加到列表控件中,可以选定需要为其指定项目的控件,然后从 Properties 窗口中选定它的 items 属性。Ellipsis 按钮将出现在属性值区的右边。单击该按钮将显示如图 9.6 所示的 String List Editor 对话框。

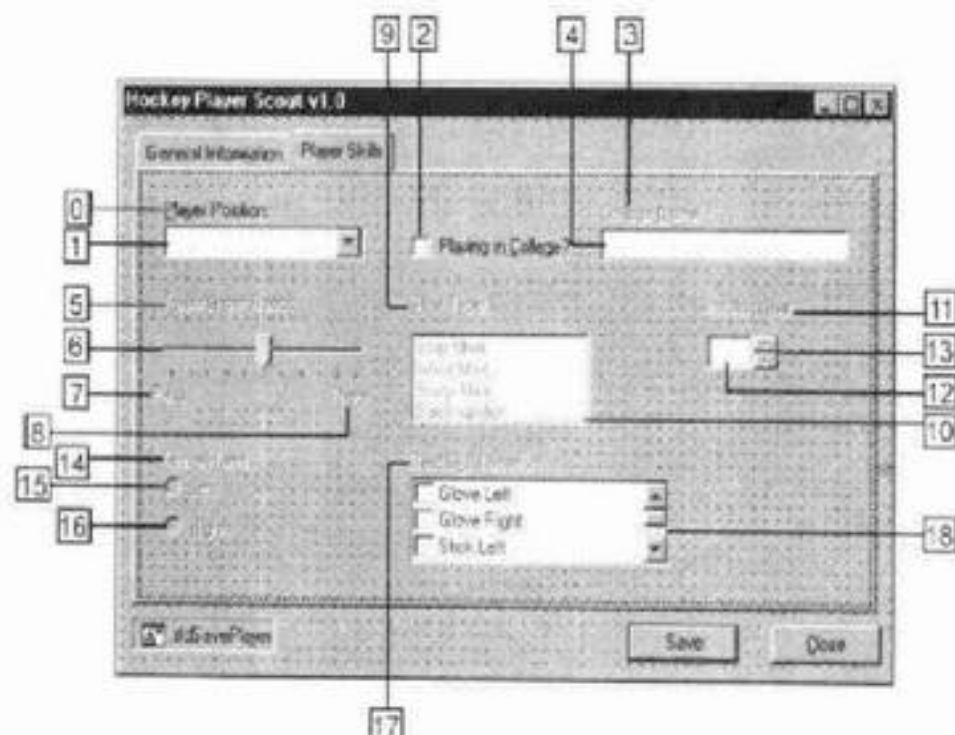


图 9.5 添加控件后的 Player Skills 选项卡

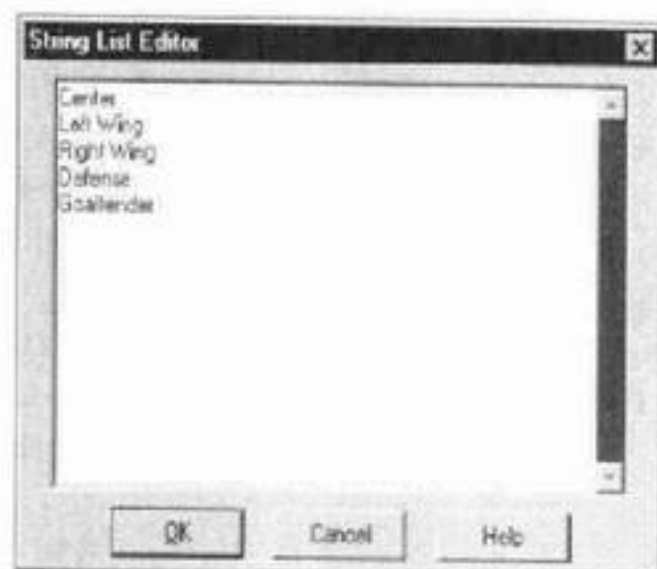


图 9.6 String List Editor 对话框

要将一个项目添加到列表中,可以键入项目条目,然后按 Enter 键。重复进行这种操作,直到添加完所有的项目,然后单击 OK。Visual J++ 将把代码添加到 initForm 方法中,以显示选定的项目。使用表 9.5 添加用来显示 cmbPosition、lstShotType、clstSkillLoc 控件的项目。

表 9.5 HockeyScout.java 列表控件项目

列表控件	选项
cmbPosition	Center
	Left Wing
	Right Wing
	Defense
lstShotType	Goaltender
	Slap Shot
	Wrist Shot
	Snap Shot
clstSkillLoc	Backhander
	Glove Left
	Glove Right
	Stick Left
	Stick Right
	Five Hole

cmbPosition 控件对于该选项卡控件的操作是至关重要的。如果您熟悉冰球,那么您也许会注意到,其中一些技术信息只适用于守门员,而有些信息只适用于其他运动员。并且,其中大多数控件都没有启用。当用户从 cmbPosition 下拉列表选定一个项目时,我们在下一步中编写的代码将确定它是否为守门员,将启用那些只适用于守门员的控件;否则,只启用适用于其他运动员的控件。将 cmbPosition 的 style 属性设置为 Dropdownlist 允许用户从列表中进行选择,但不允许在控件的编辑区进行键入。

Player Skills 选项卡使用的另一个特别的列表控件是 CheckedListBox 控件 clstSkillLoc。CheckedListBox 控件允许用户通过在每个项目旁设置一个复选标记来选定列表中的多个项目。我们将使用该功能来显示守门员可以胜任的多个位置。

chkCollege 复选框,像 cmbPosition 控件一样,也决定其他控件的状态。当选中该项目时,将启用 txtCollege 和 lblCollege。使用一个控件来设置其他控件的状态是很常见的。通过控制控件的状态,可以控制输入到应用程序中的数据流。

在添加支持该应用程序的功能的代码之前,想强调一下 Player Skills 选项卡中的另外两个控件。这两个控件提供了相同类型的信息,但形式稍有不同。

TrackBar 控件 trbPassShoot 是一个滑块控件,它使您可以像设置音响音量那样来以图形的方式设置一个整数值。Windows 在【控制面板】中设置屏幕分辨率,以及设置声卡音量时,都使用了该控件。在该示例中,我们使用它来提供一种简单的方法,以指定一个 1~10 的数。我们将使用该控件来指定前锋队员射门与传球的比值。TrackBar 控件的值是一个整数,我们可以很方便地用来确定传球与射门的比值。

UpDown 是在指定整数值时另一个特殊的控件。我们使用它来确定运动员的射门准确率。txtShotAc Edit 控件中的值越高,说明该运动员的射门准确率水平越高。我们使用 buddyControl 属性来将 UpDown 控件与 txtShotAc Edit 控件相关联,以向用户显示 UpDown 控件的设置。当用户单击 UpDown 控件的向上箭头时,Edit 控件的值将增加;当用户单击该控件的向下箭头时,Edit 控件的值将减小。这种关联就是为什么我们决定将 txtShotAc 的 readOnly 属性设置为 true 的原因。它防止用户输入任何文本,例如字符等,并使 UpDown 控件成为设置值的唯一方法。

添加支持代码

设置完控件后,下面将添加一些事件处理程序,并为应用程序编写功能代码。代码将获取用户指定的信息,然后将它编写到.ini 格式的文件中。我们将使用 Win32 API 方法 WritePrivateProfileString 来把应用程序的数据写到文件中。我们将使用 J/Direct Call Builder 添加 API 方法。编写完文件后,我们将关闭应用程序。虽然接受来自用户的数据,将它写到.ini 文件中,然后立即退出不是通常程序所要做的,但它允许我们演示一些控件的行为,存储示例的数据。

为了开始这些操作,使用表 9.6 来定义应用程序的事件处理程序。

表 9.6 HockeyScout.java 事件处理程序

控件	事件	处理程序名称
btnLoadPicture	click	LoadPicture
btnSave	click	SavePlayer
btnClose	click	ExitApp
chkCollege	checkStateChanged	CollegeCheck
cmbPosition	selectedIndexChanged	PositionChanged

现在我们已经创建了事件处理程序,下面我们将为它们添加代码,并添加一

些成员变量,以及构造函数中的 Win32 API 方法声明等。添加下面除了代码末尾的 API 声明以外的代码。要添加 API 声明,并学习如何使用 J/Direct Call Builder,您可以按照代码后的指导进行操作。



```
private String msAppName = "Hockey Player Scout";
private String ms
PicturePath = "";
public HockeyScout( )
{
    super( );
    // Required for Visual J++ Form Designer support
    initForm( );

    // Initialize DateTimePicker control to today's Date
    dtpBDay.setValue(new Time( ).getDate( ));
}

private void CollegeCheck(Object source, Event e)
{
    txtCollege.setEnabled(chkCollege.getChecked( ));

    lblCollege.setEnabled(chkCollege.getChecked( ));
}

private void SavePlayer(Object source, Event e)
{
    /* Display the SaveFileDialog object and get filename
    * to save player record to
    */
    if(sfdSavePlayer.showDialog( ) == DialogResult.OK)
    {
        // Store filename from SaveFileDialog
        String tempFile = sfdSavePlayer.GetFileName( );

        // Save the file to an .ini file format
        // Write out first name entry
        this.WritePrivateProfileString("General Information",
                                         "FirstName",
                                         txtFirstName.getText(),
                                         tempFile);

        // Write out last name entry
```

```
this.WritePrivateProfileString("General Information",
                                "LastName",
                                txtLastName.getText(),
                                tempFile);

// Write out middle name entry
this.WritePrivateProfileString("General Information",
                                "MiddleName",
                                txtMiddleName.getText(),
                                tempFile);

// Write out birthdate entry
this.WritePrivateProfileString(
    "General Information",
    "Birthdate",
    dlpBDay.getValue().formatShortDate(),
    tempFile);

// Write out birthplace entry
this.WritePrivateProfileString("General Information",
                                "Birthplace",
                                txtBirthPlace.getText(),
                                tempFile);

// Write out height entry
this.WritePrivateProfileString("General Information",
                                "Height",
                                txtHeight.getText(),
                                tempFile);

// Write out weight entry
this.WritePrivateProfileString("General Information",
                                "Weight",
                                txtWeight.getText(),
                                tempFile);

// Write out player picture entry
this.WritePrivateProfileString("General Information",
                                "PlayerPicturePath",
                                msPicturePath,
                                tempFile);

// Write out player position entry
this.WritePrivateProfileString(
    "Player Skills",
    "PlayerPosition",
    String.valueOf(cmbPosition.getSelectedIndex()),
```

```

        tempFile);
// Write out player position string
this.WritePrivateProfileString(
    "Player Skills",
    "PlayerPositionString",
    cmbPosition.getSelectedItem().toString(),
    tempFile);

// Write out college checkbox entry
this.WritePrivateProfileString(
    "Player Skills",
    "College",
    String.valueOf(chkCollege.getChecked()),
    tempFile);
if(chkCollege.getChecked())
{
    // Write out college name entry
    this.WritePrivateProfileString(
        "Player Skills",
        "CollegeName",
        txtCollege.getText(),
        tempFile);
}

// If Goalie, save just the 2 fields
if (cmbPosition.getSelectedIndex() == 4)
{
    // Write out glove hand entry
    if (optLeft.getChecked() == true)
        this.WritePrivateProfileString("Player Skills",
                                         "Glove",
                                         "Left",
                                         tempFile);
    else
        this.WritePrivateProfileString("Player Skills",
                                         "Glove",
                                         "Right",
                                         tempFile);

    // Write out skill location entries

    int[] tempSel = cistSkillLoc.getCheckedIndices();
    String tempString = new String();

    for(int i = 0; i < tempSel.length; i++)
    {

```

```

// Write entries to file
this.WritePrivateProfileString(
    "Player Skills",
    "SkillsLoc" + i,
    String.valueOf(tempSel[i]),
    tempFile);

// Write skill location strings that were selected
this.WritePrivateProfileString(
    "Player Skills",
    "SkillsLocString" + i,
    lstSkillLoc.getItem(i).toString(),
    tempFile);
}

// Write out count of skills
this.WritePrivateProfileString(
    "Player Skills",
    "SkillCount",
    String.valueOf(tempSel.length),
    tempFile);
}

else if (cmbPosition.getSelectedIndex() != -1)
{
    // Write out pass/shoot ratio entry
    this.WritePrivateProfileString(
        "Player Skills",
        "PassShootRatio",
        String.valueOf(trbPassShoot.getValue()),
        tempFile);

    // Write out shot type entry
    this.WritePrivateProfileString(
        "Player Skills",
        "ShotType",
        String.valueOf(lstShotType.getSelectedIndex()),
        tempFile);

    // Write out shot type string entry
    this.WritePrivateProfileString(
        "Player Skills",
        "ShotTypeString",
        lstShotType.getSelectedItem().toString(),
        tempFile);
}

```

```

        // Write out shot accuracy entry
        this.WritePrivateProfileString(
            "Player Skills",
            "ShotAccuracy",
            txtShotAc.getText(),
            tempFile);
    }

    MessageBox.show("The player record has been saved to "
        + "disk. The program will now close",
        msAppName,
        MessageBox.ICONINFORMATION);
    Application.exit();
}

private void ExitApp(Object source, Event e)
{
    // Check if user wants to exit the application
    if (MessageBox.show("Are you sure you want "
        + "to exit the application?",
        msAppName,
        MessageBox.YESNO
        + MessageBox.ICONQUESTION)
        == DialogResult.YES)
        Application.exit();
}

private void LoadPicture(Object source, Event e)
{
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.CheckFileExists(true);
    ofd.CheckPathExists(true);
    ofd.Filter("Bitmap Files (*.bmp)|*.bmp"
        + "|JPEG Files (*.JPG)|*.JPG");
    ofd.DefaultExt("*.bmp");

    /* Display the open file dialog.
       If user selected OK, load image file
       into PictureBox control
    */
    if (ofd.ShowDialog() == DialogResult.OK)
    {
        // load selected image file
        try

```

```

        |
        | picPicture.setImage(new Bitmap (ofd.GetFileName()));
        |
        | // Store the path to the picture
        |
        | msPicturePath = ofd.GetFileName();
        |
        | catch (Exception ex)
        |
        | /* More than likely an invalid image file has been
        |    loaded. Alert the user that the selected file
        |    cannot be loaded
        | */
        |
        | MessageBox.show("The selected file is not a valid
        |                    *
        |                    + "image file. Please check and "
        |                    + "load again.",
        |                    msAppName,
        |                    MessageBox.ICONERROR);
        |
        | return;
        |
        |
    }

private void PositionChanged(Object source, Event e)
{
    boolean isGoalie = false;
    // Check for goalie being selected.
    if (cmbPosition.getSelectedIndex() == 4)
        isGoalie = true;
    // Enable or disable goalie specific entries
    lblGloveHand.setEnabled(isGoalie);
    optLeft.setEnabled(isGoalie);
    optRight.setEnabled(isGoalie);
    lblSkillLoc.setEnabled(isGoalie);
    clstSkillLoc.setEnabled(isGoalie);
    // Enable or disable forward players.
    lblPassShoot.setEnabled(! isGoalie);
    trbPassShoot.setEnabled(! isGoalie);
    lblPass.setEnabled(! isGoalie);
    lblShoot.setEnabled(! isGoalie);
    lblShotType.setEnabled(! isGoalie);
    lstShotType.setEnabled(! isGoalie);
}

```

```

lblShotAc.setEnabled(! isGoalie);
txtShotAc.setEnabled(! isGoalie);
udShotAc.setEnabled(! isGoalie);
}

/* *
 * @dll.import("KERNEL32", auto)
 */
public static native boolean WritePrivateProfileString(
    String lpAppName,
    String lpKeyName,
    String lpString,
    String lpFileName);

```

首先,让我们来介绍一下如何将 Win32 API 声明添加到窗体的类中。我们将使用添加的 API 函数将信息保存到运动员选拔报告中。要添加任何 Win32 API 声明,您应该考虑使用 Visual J++ 附带的 J/Direct Call Builder。该生成器允许您搜索指定的 API,并将 API 方法、结构或常量的声明添加到其中一个项目源文件中。要启动 J/Direct Call Builder,可以从 View 菜单中选择 Other Windows,然后从出现的子菜单选择 J/Direct Call Builder 命令。这时将出现如图 9.7 所示的 J/Direct Call Builder 窗口。

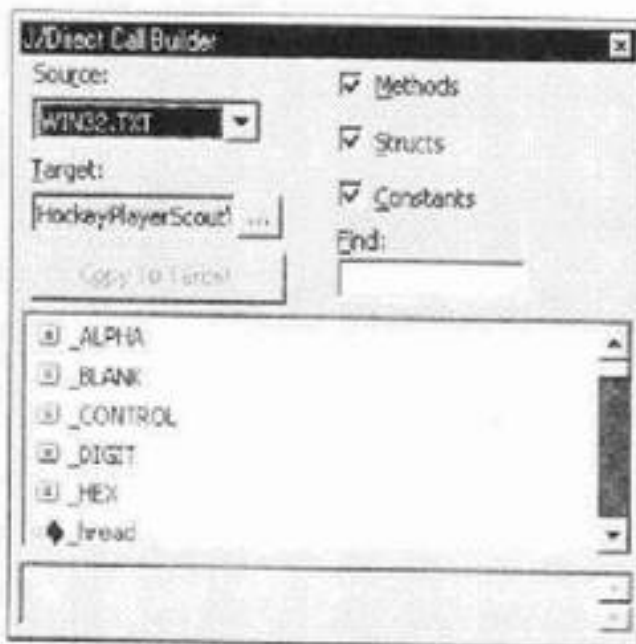


图 9.7 J/Direct Call Builder 窗口

我们需要添加的 API 是 WritePrivateProfileString 方法。它允许您创建一个 .ini 文件(纯文本)来存储设置,并被 Windows 本身用来存储操作系统的设置。该

方法将每个设置存储为一个包含键/值对的字符中。要在 J/Direct Call Builder 对话框中查找 WritePrivateProfileString 方法,可以在 Find 框中键入它的名称。当您键入时,列表将查找第一个匹配的 API 函数。查找到所需的函数后,单击 Target 框旁的 Ellipsis 按钮,选定 HockeyScout 类,然后单击 OK 按钮。这将告诉 J/Direct Call Builder,我们希望将 API 声明投递到的位置。单击 Copy To Target 按钮,J/Direct Call Builder 将把下面的代码粘贴到类声明的后面。



```

/*
 * @dll.import("kernel32", auto)
 */
public static native boolean WritePrivateProfileString(
    String lpAppName,
    String lpKeyName,
    String lpString,
    String lpFileName);

```

注释标记 @dll.import 通知编译器,下面的代码是导入的本地方法,这里是 Win32 API 方法。注释标记还指出方法的位置,这里是 Kernel32.DLL 文件。在注释标记后面,J/Direct Call Builder 声明方法。我们现在可以在应用程序的代码中使用该 API。

我们手工添加的第一段代码由两个私有成员变量 m_sAppName 和 m_sPicturePath 组成。m_sAppName 变量将存储调用 MessageBox 类时使用的应用程序的名称。m_sPicturePath 变量存储用户选定的照片文件的位置。当要保存运动员照片的位置时,SavePlayer 方法使用该变量。

我们添加的大多数代码都存在于 SavePlayer 事件处理程序中,所以让我们马上来看一看。虽然该程序很长,但它非常直观。它从调用我们添加的 SaveFileDialog 控件的 showDialog 方法开始,如果用户输入要存储的路径和文件名,那么代码将使用 WritePrivateProfileString 方法将信息写入到选定的文件中,直到它获得第二个选项卡的 chkCollege 复选框状态。代码决定复选框的状态,然后决定是否要写出学院名称。如果用户选中复选框,那么学院名称被存储;否则,它将一起被略过。

SavePlayer 存储学院信息后,代码将确定当前的运动员是否为守门员。通过检查 cmbPosition 控件的值,并确定控件选定的索引是否等于 4(守门员项目的索引)。如果是守门员,那么代码不需要存储关于射门方面或射门准确率的数据,所以它将略过这些控件,然后写出守门员特定的信息。

在守门员区中,代码写出 clstSkillLoc 控件的选定内容。因为用户可以选定多个项目,所以代码需要能够在 .ini 文件中处理多个条目。要使其他应用程序能够更容易地读取值,代码将列表控件中每个选定的项目在 .ini 文件中存储为

两个独立的条目。第一个条目的值是列表控件中选定项目的索引。另一个条目的值选定项目的技术位置字符串。代码使用 `getCheckedIndices` 方法来统计选定项目的数量,然后在 .ini 文件中循环处理每个条目。列表控件选定项目索引的条目键以“SkillLoc”开始,以循环索引结束。技术位置字符串条目键以相同的方式建立,但是以“SkillsLocString”开始。因此, `clstSkillLoc` 中的一个选定项目也许以下面的方式保存:

```
SkillLoc2=3  
SkillsLocString2=Stick
```

`LeftSavePlayer` 方法也将一些选定的项目写入到文件中,以便读回信息更加容易。

如果选定的运动员位置不是守门员,那么代码将只把前锋队员控件信息写出到文件中。代码将检查值是否为 -1 (它表示用户没有选定位置)。在控件返回整数值的情况下,例如 `TrackBar` 和 `ListBox` 控件, `String` 对象的 `valueOf` 方法将进行转化。代码完成将信息写到文件中时,将显示一个消息框,告诉用户文件被保存,将要退出应用程序。然后代码使应用程序终止。

列表控件将索引同时写到选定项目和字符串的原因是使读取信息尽可能得灵活。如果只想将信息读回到同一个应用程序中,那么可以只读取索引,然后在指定的列表控件中选定适当的项目。另一方面,如果需要将项目添加到列表中,那么也许在程序将数据读回到应用程序中以无效的索引结束。具有指定索引的项目也许在列表中的位置不同。要解决这个问题,可以读回字符串,然后将它与要选定的列表项目进行比较。如果它们不匹配,那么可以采取适当的措施。通过将字符串写到文件中,还可以使其他应用程序只读取字符串,而不必考虑了解它们在原始列表中的索引顺序。当编写一个从该应用程序读取信息的示例时,将遇到这种情况。

当用户单击 `btnLoadPicture` 命令按钮控件时,该命令按钮控件的 `click` 事件调用 `LoadPicture` 方法。`LoadPicture` 的代码创建和初始化 `OpenFileDialog` 对象,并设置其选项和筛选开始。允许使用的照片文件类型只有 JPEG 和标准的 Windows 位图。如果用户选定一个图像文件,那么代码将试着根据用户选定的图像设置 `PictureBox` 控件的 `image` 属性。代码还将在 `m_sPicturePath` 成员变量(在前面的步骤中提到过)中存储到该文件的路径。

我们已经编写了将图像分配给尝试/捕获块中的 `PictureBox` 的代码,以确保在分配图像时不会有意外发生。为了防止应用程序在用户选定无效文件时失败,应该允许用户打开文件,并将它直接加载到控件中时,我们通常使用尝试/捕获块。在捕获块中,代码只显示一个消息框,警告用户这种意外的一般原因,然

后退出事件处理程序。

最简单的事件处理程序是 CollegeCheck 处理程序。这种方法简单地将 txtCollege 和 lblCollege 控件的 enable 属性设置为 chkCollege 选中状态的值。类似地, PositionChanged 事件处理程序根据 cmbPosition 列表中选定的内容来启用或禁用控件。代码将创建一个名为“isGoalie”的 boolean 变量。我们将使用该变量来设置 Player Skills 选项卡页面中的控件。如果 cmbPosition 控件当前选定项目的索引是 4(守门员项目),那么代码将把 isGoalie 变量设置为 true;否则,将它的创建状态默认为 false。然后代码继续将面向守门员控件(如 clstSkillLoc 和 optGoalie)的 enabled 属性设置为 isGoalie 的值,并将非守门员字段设置为 isGoalie 相反的值。该方法中的代码允许用户只访问对于指定运动员位置所需的控件。

现在我们可以编译并运行应用程序了。一旦运行该应用程序后,就可以创建运动员记录了。注意,根据您选定的内容,控件将具有不同的启用状态。当输入数据时,标号顺序(在将控件添加到窗体时指定)将使在创建运动员记录时切换控件更加容易。

在创建应用程序时,如果标号顺序不是您所希望的,那么可以使用图 9.4 和图 9.5 来查看指定标号的方法。使用 Tab Order 视图来修改设置(当显示 Forms Designer 时,通过从 View 菜单中选择 Tab Order,可以选择这些设置)。创建完这些记录后,在编辑器中打开文件,然后查看存储的数据。要特别注意守门员技术位置在文件中的存储方式。

现在您应该理解如何使用基本控件了。我们没有使用 Visual J++ 提供的所有控件,而只是使用常见的控件。要了解关于基本控件的属性、方法和事件方面的内容,我们建议您使用控件类的 WFC 类引用。

9.2 在应用程序中加入帮助

无论应用程序的用户界面设计得多好,操作多么简单,您都应该提供帮助,因为这样可以减少客户支持的问题,并使应用程序能够更好地适应您的工作。Visual J++ 提供了一些向用户显示帮助信息的方法。从 HTMLHelp、WinHelp 文件,到“*What's This*”类型的帮助,Visual J++ 都提供了一些使这种操作更加简单的工具。

在本节中我们通过 9.1 节的例子介绍如何将帮助添加到应用程序中。我们将使用 com.ms.wfc.ui.Help 类来显示对话框的帮助主题。还将使用 WFC 的 HelpProvider 控件来将帮助与应用程序第一个选项卡中的每个控件联系起来,并显示应用程序第二个选项卡中的每个控件的弹出式或“*What's This*”帮助。我们

将在该示例中使用 Microsoft 新的帮助模式: HTMLHelp。(对于大多数情况,我们这里使用的代码也都适用于显示 WinHelp 文件。)完成该示例后,您将能够将 HTMLHelp 合并到应用程序中。

9.2.1 加入帮助前的准备

本小节的完整例子在随书光盘的 samples\chap09\HockeyPlayerScoutHelp 目录下。将本示例中使用的 HTML 帮助文件添加到 HockeyPlayerScoutHelp 目录中。帮助文件的名称为 HockeyScout.chm, 必须将文件放到 HockeyScoutHelp(您的工程目录)中, 可以用系统目录下的 hh.exe 打开此帮助, 以 HockeyScout.chm 为命令行参数。要了解 HTMLHelp 详细信息, 请参见 HTMLHelp Workshop 文档的有关内容。

9.2.2 打开 HockeyPlayerScout 应用程序并添加帮助控件

打开 HockeyPlayerScoutHelp 下的 HockeyPlayerScout.vjp 工程, 显示工程后, 在 Forms Designer 中打开 HockeyScout.java 文件。窗体如图 9.8 所示。

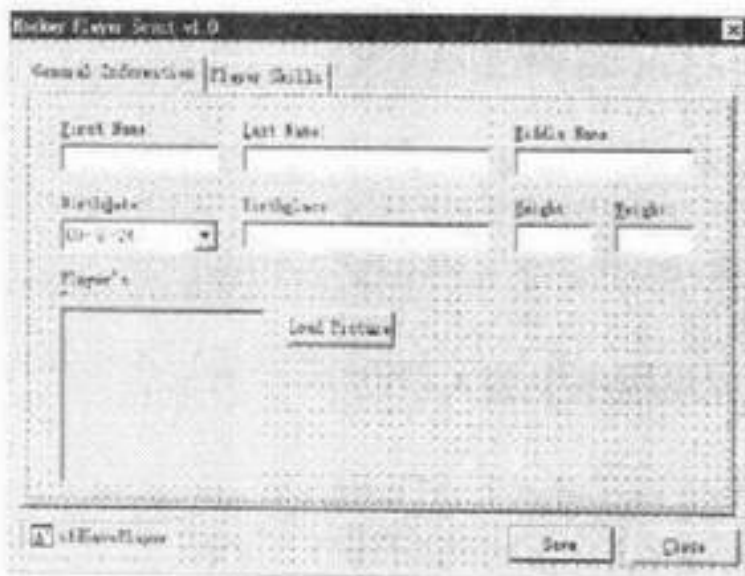


图 9.8 添加帮助控件前的 HockeyScout.java 文件

我们需要将两个控件添加到窗体中。首先添加 Button 控件, 然后将它放到窗体 Save 按钮的旁边。将该按钮命名为“btnHelp”, 并将其 text 属性设置为“Help”。接着将 HelpProvider 控件放在窗体 Tab 控件下的任意位置。该控件不提供用户界面, 所以放置位置不是很重要。将 HelpProvider 控件命名为“hlpMain”, 并将其 helpFile 属性设置为“HockeyScout.chm”。图 9.9 显示了添加控件后的

HockeyScout 窗体外观。

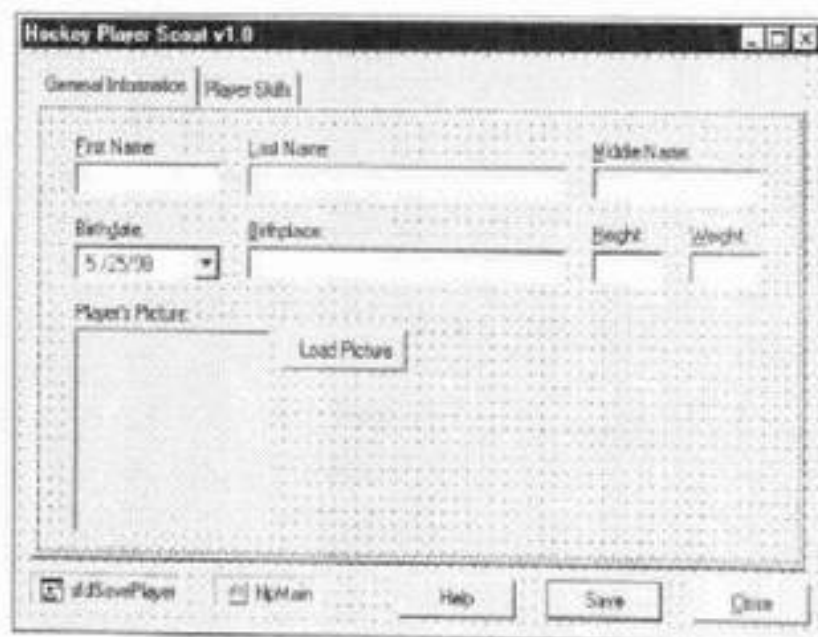


图 9.9 添加帮助控件后的 HockeyScout.java 文件

HelpProvider 控件是一个派生控件。派生控件扩展其他组件的功能属性。HelpProvider 控件将添加一个属性,该属性允许您指定与控件和窗体相关的帮助主题或帮助字符串。在该示例中,我们将不使用这些属性,因为理解如何使用代码来设置帮助系统更加重要。为了在对话框的标题中显示“What's This”帮助,我们需要将窗体的 helpButton 属性设置为 true。这样做后,您将会看到窗体的标题区、Close 按钮旁添加了一个问号按钮。稍后在该示例中,我们将把代码添加到 HelpProvider 控件中,来指定当用户单击“What's This”帮助按钮,然后单击 Player Skills 选项卡页面中的控件时显示的帮助字符串。

9.2.3 添加 Help 按钮的事件处理程序及代码

现在我们已经添加了控件,下面将创建 Help 按钮的事件处理程序。在帮助文件中,有两个主题描述当前选定的选项卡页面的作用。Help 按钮将显示与当前选项卡作用相关的主题。添加 btnHelp click 事件的处理程序,然后输入下面的代码:



```
private void DisplayHelp(Object source, Event e)
{
    // Determine which tab is selected
```



```

        if (tabMain.getSelectedIndex() == 0)
        {
            // Display the topic about the general tab
            Help.showHelp(this, "HockeyScout.chm", "default.htm");
        }
        else
        {
            // Display the topic about the skills tab
            Help.showHelp(this, "HockeyScout.chm", "hockey.htm");
        }
    }
}

```

代码开始确定当前选定的选项卡,以便当用户单击 Help 按钮时显示适当的主题。然后代码调用 Help 类的 showHelp 方法,它指出了指定的帮助主题。该方法需要以下参数:要作为帮助请求源的控件或窗体、帮助文件的名称,以及要显示的主题。HTML 帮助文件中每个主题都是一个独立的 HTML 文件,所以我们需要在主题名称中使用 .htm 扩展名。

9.2.4 添加支持 F1 键和“What's This”帮助的代码

前面介绍了如何使用 com.ms.wfc.ui.Help 类来显示将整个对话框或选定的选项卡文档化的主题。您还可以为每个控件添加帮助。当应用程序的控件难于理解时,通常会提供这种帮助。如果决定提供控件的帮助(或者称为“F1 帮助”),那么您可以使用 HTMLHelp 来显示当用户选定它们,然后按 F1 键时的控件主题。

“What's This”类型的帮助通过单击对话框标题栏中的“What's This”按钮(问号按钮),然后单击需要帮助的控件来显示。如果控件比较简单,那么这种帮助很有用,但是因为对话框中的控件通常是集成的,所以最好使用介绍整个对话框的帮助主题,而不是使用描述单个控件的小文本字符串。

下面我们将这两种帮助添加到应用程序中。对于 General Information 选项卡,我们将把选项卡中所有的控件添加到 default.htm 主题中。第二个选项卡将使用“What's This”帮助来显示每人数据条目控件的简要描述。因为在启动应用程序时将调用我们添加的代码,所以我们需要对窗体的构造函数进行一些修改。将下面的代码添加到 HockeyScout 窗体中:



```

public HockeyScout()
{
    super();
}

```

```

//Required for Visual J++ Form Designer support
initForm( );

// Initialize DateTimePicker control to today's Date
dtpDay.setValue(new Time( ).getDate( ));

// Get topics and help strings wired up
SetupHelp( );
!

/* * Wires up F1 help for controls on first tab and help strings
 * on second tab
 */
private void SetupHelp( )
{
    // Setup help for first tab
    Control[] tabCtrls = tpGeneral.getControls( );
    for (int x = 0; x < tpGeneral.getControlCount( ); x++)
    {
        hlpMain.setHelpTopic(tabCtrls[x], "default.htm");
    }

    // Setup help strings for second tab
    hlpMain.setHelpString(cmbPosition,
        "The position in hockey "
        + "for the player.");
    hlpMain.setHelpString(chkCollege,
        "Did this player play in college?");
    hlpMain.setHelpString(txtCollege,
        "The name of the college that "
        + "the player attended.");
    hlpMain.setHelpString(trbPassShoot,
        "The player's tendency "
        + "to shoot or pass.");
    hlpMain.setHelpString(lstShotType,
        "The player's best type of shot.");
    hlpMain.setHelpString(txtShoAcc,
        "The accuracy level "
        + "when the player shoots.");
    hlpMain.setHelpString(optLeft,
        "The goaltender's catching glove is "
        + "located in his left hand");
    hlpMain.setHelpString(optRight,
        "The goaltender's catching glove "
        + "is located in his right hand");
}

```



```

hipMain.setHelpString(cistSkillLoc,
    "The goaltender's best "
    + "save locations.");

```

我们将 SetupHelp 方法的调用添加到 HockeyScout.java 的构造函数中。SetupHelp 方法的代码以创建 tpGeneral 选项卡页面上的控件引用阵列开始。然后代码开始循环,并调用 HelpProvider 的 setHelpTopic 方法来将阵列中的每个控件与 default.htm 主题联系起来。HelpProvider 控件管理这种关系,并在用户选定控件后按 F1 键时调用帮助文件。然后代码继续将帮助字符串与 Player Skills 选项卡页面上的控件联系起来。setHelpString 方法,像 setHelpTopic 方法那样,将控件与相关的帮助联系起来。其中的一个不同是只使用一个文本字符串而不是 HTML 帮助主题来作为帮助信息。这两种帮助的另一不同是 General Information 选项卡要求用户选定控件,然后按 F1 键来显示帮助。要显示关于 Player Skills 选项卡的帮助,用户可以单击对话框标题栏上的“What's This”按钮,然后选择选项卡中需要帮助的控件。如果没有指定 Player Skills 选项卡页面控件的帮助字符串,那么当用户在控件上单击“What's This”按钮时,将显示默认的主题。

9.2.5 运行结果

添加完代码后,我们的示例也就完成并可以运行了。编译并运行项目,如图 9.10 所示。

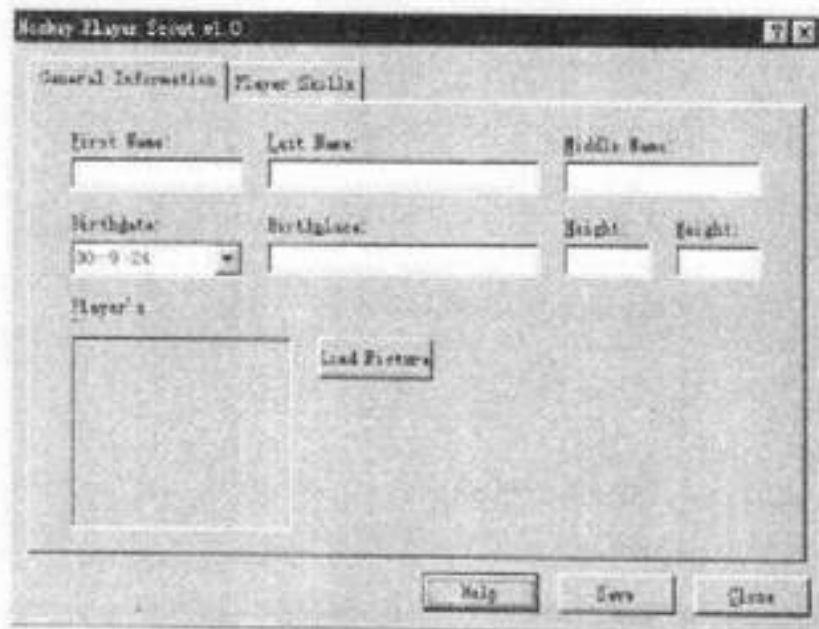


图 9.10 运行结果

当应用程序出现时,单击 **Help** 按钮,将会看到 **General Information** 选项卡的帮助主题,如图 9.11 所示。

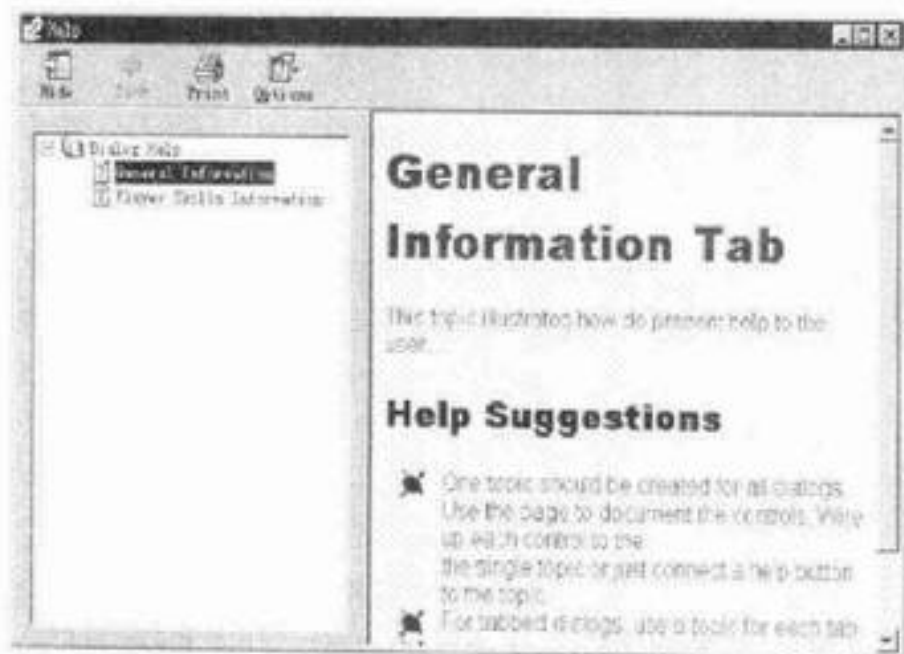


图 9.11 General Information 选项卡的帮助主题

选中 **Player Skills Information** 选项卡,然后单击 **Help** 按钮,将会看到该选项卡的帮助主题,如图 9.12 所示。

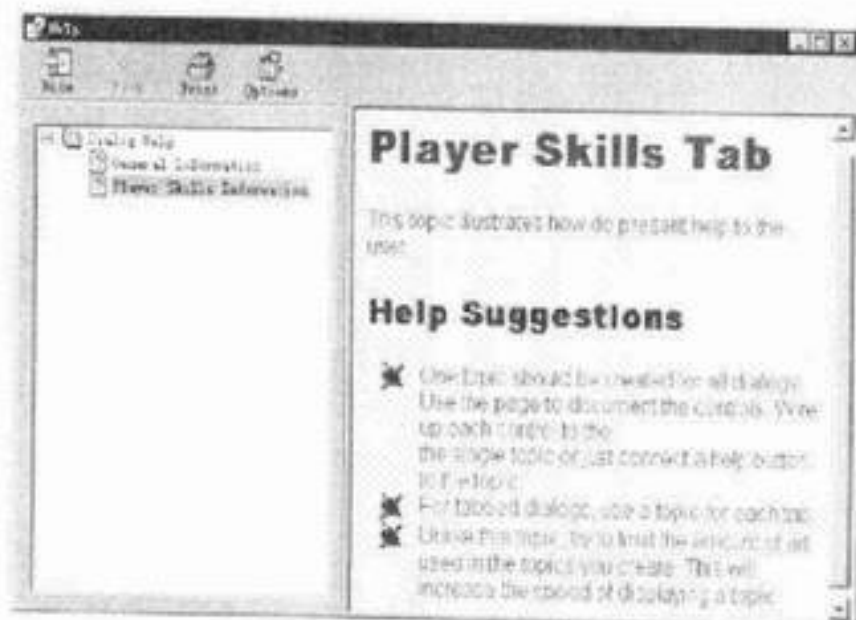


图 9.12 Player Skills Information 选项卡的帮助主题

选择对话框标题栏上的“What's This”帮助按钮,然后单击 Player Skills Information 选项卡中的一个按钮,将会看到一个描述该控件的弹出式窗口。切换回 General Information 选项卡,然后按 F1 键,将看到与单击 Help 按钮时相同的主题。

9.3 在应用程序中支持拖放

Windows 有一个使它不同于少数操作系统的功能,它具有从一个控件将信息拖放到另一个控件的能力。这种功能就是我们所说的“拖放”,它可以使一个好的应用程序功能变得更加完美。例如,一个顺序输入应用程序允许用户从数据库中删除事务,可以通过允许用户将事务从数据库拖到用于删除文件的回收站中来简化这个操作。

下面,我们将创建一个允许打开多个文本文件并将它们存储在一个列表中的应用程序。从列表中,可以拖动文件,然后将它们放到用于查看的编辑控件中。图 9.13 显示了该应用程序运行时的外观。我们的目的是展示拖放在 WFC 中的工作方式,并演示如何在自己的项目中使用它。

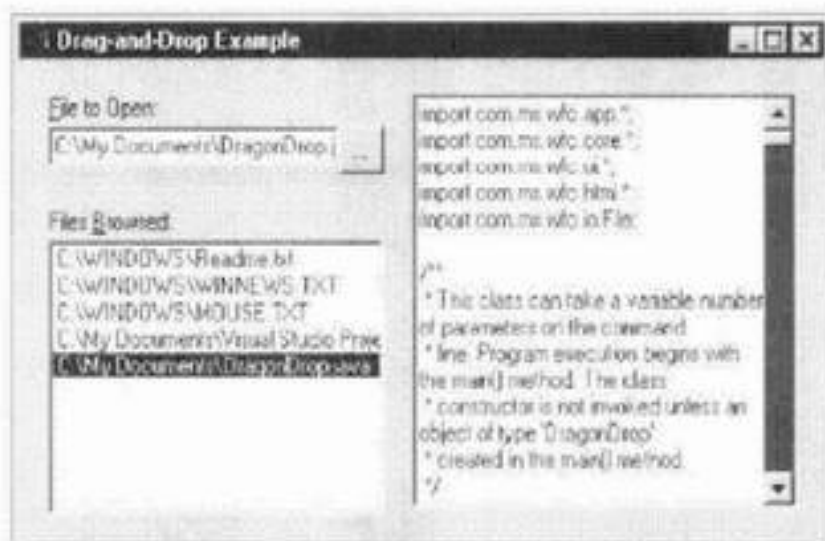


图 9.13 运行时的 DragonDrop 应用程序

9.3.1 创建工程及其窗体

使用 Empty Project 模板创建一个新工程,并将其命名为“DragonDrop”。将一

一个新窗体添加到工程中,并将其命名为“DragonDrop.java”。DragonDrop 窗体有一个添加到列表中最后一个文件的 Edit 控件、显示 OpenFileDialog 对话框的 Button 控件、存储浏览过的文件的 ListBox 控件,以及查看被拖放到其中的文件的多行 Edit 控件。使用表 9.7,将这些控件添加到窗体中,并指定其属性设置。

表 9.7 DragonDrop.java 组件及属性设置

组件类型/ID	属性	设置
Form	size.x	439
	size.y	274
	startPosition	Center Screen
	text	Drag-and-Drop Example
Label - 0	name	lblFileToOpen
Edit - 1	text	&File to Open:
	name	txtFile
	text	""
Button - 2	name	btnBrowse
	text	...
Label - 3	name	lblBFiles
	text	Files &Browsed:
ListBox - 4	name	lstBFiles
Edit - 5	name	txtDocument
	allowDrop	true
	anchor	All
	text	""
	multiline	true
	scrollBars	Vertical
OpenFileDialog	name ofdMain	
	defaultExt	.txt
	filename	*.txt
	filter	Text Files (*.txt) *.txt All Files (*,*) *,*



注意 表中为每个组件指定的 ID 号与该控件的 tabIndex 值标识了图 9.14 中的控件。(OpenFileDialog 控件没有 tabIndex 属性。)

图 9.14 显示了添加完控件并指定其属性后的窗体外观。

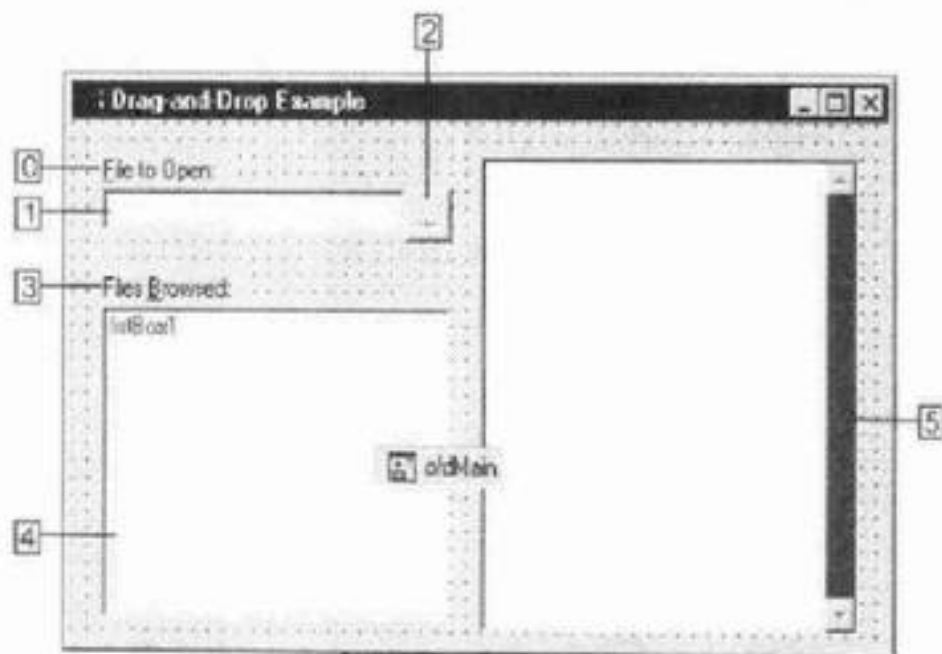


图 9.14 设计时的 DragonDrop.java

将 `txtDocument` 的 `anchor` 属性设置为 `All`, 这将会调整控件的大小, 以便在调整窗口大小时它的边界保持与窗口边界的距离。

9.3.2 创建窗体控件的事件处理程序

设置完控件后, 下面我们来创建 3 个事件处理程序。第一个是为 `btnBrowse` 的 `click` 事件创建的。我们将使用该事件处理程序来显示 `OpenFileDialog` 对话框, 以搜索要查看的文件, 并将其添加到列表框中。将按钮的事件处理程序命名为“Browse”。第二个事件处理程序是为 `lstBFile` 的 `mouseDown` 事件创建的, 将其命名为“DragFromList”。我们将使用 `DragFromList` 来初始化 `ListBox` 控件与多行 `Edit` 控件 `txtDocument` 之间的拖放操作。我们要创建的第三个事件处理程序是为 `txtDocument` 的 `dragDrop` 事件创建的, 将其命名为“DocumentDrop”。它提供我们的文件根据从 `ListBox` 控件拖放的文件名打开的代码的位置。`dragDrop` 将 `DragEvent` 对象作为它的一个参数来提供。`DragEvent` 对象提供被拖放的数据, 并允许您确定数据的格式。

9.3.3 添加拖放支持代码

将下面的代码添加到 `DragonDrop.java` 文件的开始处:

```
import com.ms.wfc.io.File;
```

现在将下面的代码添加到前面创建的事件处理程序中：



```
/* * Drop event for the txtDocument edit control.
 * Performs opening of selected file from drag
 * and drop operations.
 */
private void DocumentDrop(Object source, DragEvent e)
{
    // Create temporary DataObject to work with
    DataObject tempData = (DataObject)e.data;

    // Determine if the data being dragged is text
    if(tempData.getDataPresent(String.class))
    {
        // Open the file that was selected
        File tempFile =
            File.openRead((String)
                tempData.getData(String.class));
        // Clear the contents of the edit control
        txtDocument.clear( );
        // Read in the contents of the file
        txtDocument.setText(tempFile.readStringCharsAnsi(
            (int)tempFile.getLength( ));
    }
}

/* * Button click event for browse button. Looks up files
 * to add to list of files to browse
 */
private void Browse(Object source, Event e)
{
    // Display an OpenFileDialog to pick files to browse
    if(ofdMain.ShowDialog( ) == DialogResult.OK)
    {
        // Update the text display for the current file
        txtFile.setText(ofdMain.GetFileName( ));
        // Add the file that was selected to the list
        lstBFiles.AddItem(ofdMain.GetFileName( ));
    }
}
```


将文件的整个内容作为一个 String 对象返回)来将 txtDocument 的 text 属性设置为文件的内容。

9.3.4 拖放例子运行结果

现在我们将编译并运行该示例,结果如图 9.15 所示。

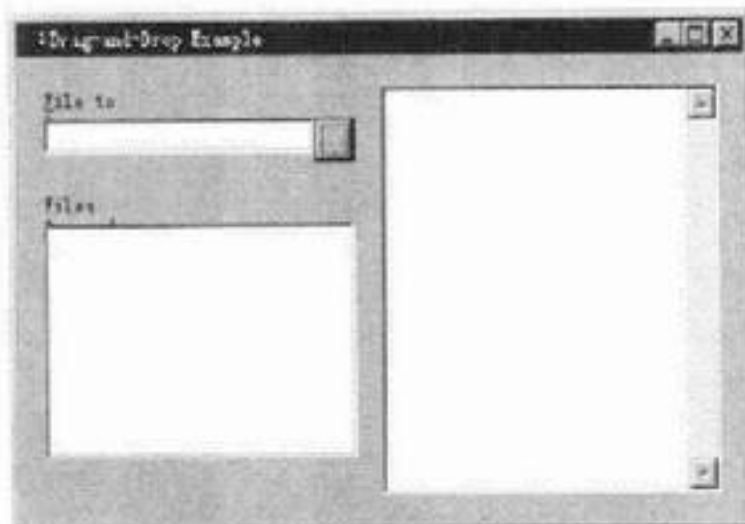


图 9.15 拖放例子运行结果

显示应用程序后,单击[...]按钮,选择要查看的文本文件。重复这个过程,直到在 lstBFiles 中列出一些文件。单击并按住 lstBFiles 中列出的一个文件,然后将它拖到 txtDocument 上。当单击项目并将它移开 ListBox 控件时,您将会注意到鼠标的变化,说明拖放操作已经开始。在 txtDocument 上释放鼠标键,文件将被加载并显示在控件中。

虽然该示例比较简单,但是代码与在应用程序中执行的其他类型的拖放操作的代码类似。

9.3.5 拖放操作的改进

您可以做一些别的事情来使应用程序支持更加强大、友好的拖放操作。使用以下这些建议来改进该示例:

- ◆ 示例将说明如何读取字符串形式的信息。修改示例使用 RichEdit 控件,以便可以拖放要查看的 RTF 及文本文件。
- ◆ 如果想将文件从【Windows 资源管理器】而不是从 ListBox 控件拖到 txtDocu-

ment 时,该怎么办? 可以将代码添加到 DocumentDrop 事件处理程序中。使用以下代码片断作为示例:



```
if (tempData.getDataPresent(DataFormats.CFHDROP)) {
    // Store files in an array in case multiple files are dropped
    String[] files = (String[])tempData
        .getData(DataFormats.CFHDROP);
    /* Since the example supports single files,
     * read only the first file
     */
    File tempFile = new File(files[0]);
    // Open the file and assign to edit control...
}
```

该代码使用包含所有“剪贴板”的枚举值及拖放对象类型的 `com.ms.wfc.app.DataFormats` 类。如果被拖放的数据是 `CF_HDROP` 格式(对于从 Windows 拖放的文件有效的格式),那么 `String` 阵列将读取并存储数据(以免拖放多个文件)。然后可以创建一个使用 `String` 阵列的第一个元素值打开的 `File` 对象。

9.4 在应用程序中使用 ActiveX 控件

随着应用程序的功能越来越丰富,它们的要求也变得越来越多。您也许会发现需要自定义用户界面控件。确实,您可以自定义现有的控件或编写自己的 WFC 控件来处理这种情况,但是这很浪费时间。如果您有指定的用户界面要求,那么别人也有可能需要这种功能。正如前面提到的,Visual Basic 和 Visual C++ 应用程序都已经使用 ActiveX 控件提供自定义功能和用户界面元素功能。HTML 页面也经常使用它们来提供比标准 HTML 能够提供的更好的图形控件。

Visual J++ 开发小组意识到 ActiveX 控件在开发实际应用程序中的重要性,所以设计 WFC 来使开发人员能够在 WFC 应用程序中使用 ActiveX 控件。当 ActiveX 控件被导入到 WFC 应用程序中时,Visual J++ 读取 ActiveX 控件的类型库,并创建控件中包含的类的封装体。Visual J++ 将控件放到应用程序内的数据包中,使您能够在项目中更容易地使用它们。

在本节中,我们将首先了解什么是 ActiveX 和 ActiveX 控件,然后通过例子说明把 ActiveX 控件导入到 WFC 应用程序中的过程。

9.4.1 ActiveX 技术背景

ActiveX 是一种技术集合,它使得在 WWW 网上交互内容很容易实现。利用 ActiveX,提供多媒体效果、交互式对象和复杂的应用程序,使网上应用变得生动活泼。

ActiveX 技术提供了一种把所有使网络生动起来的技术结合在一起的技术,它是由 Microsoft 公司定义的用于 Internet 的一种对象链接与嵌入技术(OLE)。ActiveX 是 OLE 的超集,ActiveX 包括了建立在 COM 上的 Windows 特性,而且还有 Microsoft Internet Information Server 家族和 WinInet 编程接口。术语的变化和技术的发展一样快,让我们来了解一下 ActiveX 技术的历史与发展。

我们一定做过这样的事,在 Windows 的记事本中进行录入,通过鼠标或键盘选择一部分文字,并将它们复制到剪贴板中,然后切换到 Microsoft Word 中,使用粘贴功能将剪贴板中的内容粘贴到 Word 文档中,同样,也可以在 Word 中选择一部分文字复制到剪贴板中,并将它们粘贴到记事本中。

在剪贴板引入后不久,人们又希望能有一种无需手工剪切和粘贴就可以实现在应用程序中进行沟通的方法。为了满足这方面的需求,Microsoft 公司引入了动态数据交换(DDE)技术,通过 DDE,程序员编写代码使两个应用程序之间可以进行通信。在使用 DDE 的两个应用程序中,其中一个为信息接收方,称为客户端;另一方为信息提供方,称为服务器端。虽然 DDE 在两个应用程序之间提供了强有力的通信协议,但是,大多数程序员还是发现编写和调试基于 DDE 的程序是一件相当困难的事情。

DDE 可以解决问题,由于本身的限制,不能很好地解决应用程序间交换信息的需求。Microsoft 又开发了对象链接与嵌入(OLE)技术,它使用了一种“以文档为中心”的计算模型,一种将注意力集中于文档类型而不是创建此文档的应用程序的计算模型。使应用程序可以使用其他程序的功能。例如,通过 OLE 可以在 Microsoft Word 文档中插入一个画图图像,并可以使用画图中的所有功能,也就是实地激活。如果在编辑 Word 文档中需要改变其中的画图图像,您只需在图像上用鼠标双击,Windows 就会启动画图程序,并调入此图像,在完成了对图像的修改之后,退出画图并返回到 Word 中。“以文档为中心”使各种应用程序能联合起来,共同为文档服务,而对于文档来说,它是一个整体,并不需要分开来编辑各个部分。

在使用 OLE 时,既可以在文档中嵌入对象,也可以在其中链接对象。当嵌入对象时,Windows 在文档中存放此对象的副本,如果以后在文档外对此对象进行了修改,文档中的对象将不受到影响。而如果是在文档中链接对象,Windows

只在文档中存放一个对象磁盘文件的索引,每次打开此文档时,Windows 都会将此对象相应的文件载入到文档中。这样,如果修改了此对象,在文档中都将保持一个此对象的最新内容。OLE 容器是一个嵌入或链接了一个对象的应用程序,而 OLE 服务器则是创建此被链接或被嵌入对象的应用程序。利用 OLE 技术我们可以完成许多功能强大的应用程序,其中利用 OCX 控件是最方便最简捷的。

随着 Internet 的迅速发展与普及,开发商们都把侧重点偏向了网络。以前的 OCX 文件都是很大的文件,其中可能包含了一些超出应用程序需要的功能。因此,Microsoft 开发出一种新型的对象技术,ActiveX 是一种网络定义的 OLE,ActiveX 控件就是网络上的 OLE 控件,像 OCX 一样,可以由计算机软件生产商提供,完成网络上许多复杂的功能。

可以说,ActiveX 是在“老”的 OLE 与 Internet 发生碰撞建立起来的技术。ActiveX 控件提供特定功能的二进制对象,它们可以被加入到客户/服务器应用程序、Web 应用程序、Java 小程序以及使用 JavaScript 和 VBScript 的 Web 页中。可以通过 Visual Basic、Visual C++、Borland C++ 等程序语言来创建 ActiveX 控件。ActiveX 控件的一个例子就是 Microsoft 的 Chart 控件和日历控件。使用 Chart 控件,可以画出各种具有不同风格的图表,如果需要在程序中制作图表,只需要将 Chart 控件加入到程序中就行了。此外,Chart 控件还能被用于 JavaScript 或 VBScript 写成的 Web 应用程序中,Chart 控件可以在 Microsoft 的 Web 站点上找到。

9.4.2 NumText 控件说明

我们将看一看如何与控件进行交互。我们将创建一个简单的应用程序登录对话框。图 9.16 显示了该对话框运行时的情况。



图 9.16 运行时的 ActiveXPassword 示例

为了协助处理由一些数字组成的密码(如我们的应用程序要求的),我们将合并 Visual Basic 中设计的一个 ActiveX 控件。该控件是简单的,限制只能输入数字的 Edit 控件的修订版,并允许使用 Backspace 键。



注意 为了获得在 Visual Basic 中创建的 ActiveX 控件的最佳性能,建议您使用 Visual Basic 6.0 来创建 Visual Basic 控件。该版本提供了多线程的支持,这将使 Visual J++ 和 WFC 能够正确地处理控件。没有多线程支持的旧版本的 Visual Basic 控件也许不能在 WFC 应用程序中正常使用。

9.4.3 创建工程并设计窗体

在 Visual J++ 中使用 Empty Project 模板创建一个新工程,将其命名为“ActiveXPassword”。将一个新窗体添加到该工程中,然后将其命名为“ActiveXPassword.java”。在导入 ActiveX 控件之前,我们需要设计窗体剩余的外观。使用表 9.8,将所需的控件添加到窗体中,并设置控件及属性。

表 9.8 ActiveXPassword.java 组件及属性设置

组件类型	属性	设置
Form	size.x	273
	size.y	198
	startPosition	Center Screen
	text	Please Login...
	acceptButton	btnOK
	cancelButton	btnCancel
	borderStyle	Fixed Dialog
	minimizeBox	false
	maximizeBox	false
	controlBox	false
Label	name	lblWelcome
	text	Please enter your user name and password to access this application:
Label	name	lblUsername
	text	User name:
Edit	name	txtUsername
	text	

续表

组件类型	属性	设置
Label	name	lblPassword
	text	Password:
Button	name	BtnOK
	text	&OK
Button	name	BtnCancel
	text	&Cancel

图 9.17 显示了添加所有控件后的窗体外观。



图 9.17 设计时的 ActiveXPassword.java 窗体

9.4.4 添加 ActiveX 控件

接下来我们需要将 ActiveX 控件 NumText 添加到项目及窗体中。NumText 控件包含在本书附带的光盘中。在可以将该控件导入到项目之前,您需要在注册表中注册所有的 ActiveX 控件。要注册 NumText 控件,您可以从 Windows 的【开始】菜单中选择【运行】,然后输入“RegSvr32.exe < path > NumText.ocx”,其中 < path > 指的是控件的路径。如果控件注册成功,那么将会出现一个消息框通知注册成功。

要将 NumText 控件导入到我们的工程中,您可以从 Tools 菜单中选择 Customize Toolbox 命令。这时,将出现如图 9.18 所示的 Customize Toolbox 对话框。该对话框显示所有 ActiveX 控件的列表,并提供了添加 WFC 控件和小程序的能力。



图 9.18 Customize Toolbox 对话框

要选中 NumText 控件, 查找名为“VJDEVWORKSHOP.NumbText”的控件, 选中该选项。单击 OK 按钮, 退出该对话框。现在您将看到一个新代表 NumText 控件的控件被添加到当前的 Toolbox 窗格中。单击 NumText 控件, 然后将它拖到窗体中, 以便它出现在 Password 标签下面。

当您把控件拖到窗体上时, Visual J++ 将终止并创建文件的类封装体。完成后, 您不仅能够看到窗体上的控件, 而且还可以看到 Visual J++ 将新的数据包添加到 Project Explorer 窗口的工程中。这个新的数据包包含控件所有的类封装体。将控件的名称设置为“ntxtPassword”, 将它的 passwordChar 属性设置为“*”, 将 tabIndex 属性设置为“4”, 以便用户可以从 txtUsername 转到 ntxtPassword。

9.4.5 添加事件处理程序及代码

完成用户界面后, 下面我们将添加支持代码。该应用程序的目标是检查用户的名称及密码是否有效。如果这两项都是有效的, 那么代码将向您显示一个通知消息框; 否则, 您将得到一个提示错误的消息。

在添加所需的代码之前, 我们必须为窗体中两个命令按钮的 click 事件创建处理程序。对于 btnOK 控件, 将它的 click 事件处理程序命名为“CheckPassword”。将 btnCancel 的 click 事件处理程序命名为“CancelPassword”。然后将下面的代码添加到事件处理程序及代码中显示的私有成员变量中。



```
// Stores the valid password
private int mIPassword = 9876;
```



```
private void CheckPassword(Object source, Event e)
{
    // Check to make sure the user entered a valid user name
    if (txtUsername.getText().equals(""))
    {
        MessageBox.show("Invalid user name!",
                        "Login Dialog",
                        MessageBox.ICONERROR);
        txtUsername.focus();
        return;
    }
    // Check for password being valid
    if (ntxtPassword.getNumFromText() != m_iPassword)
    {
        MessageBox.show("Incorrect password!",
                        "Login Dialog",
                        MessageBox.ICONERROR);
        ntxtPassword.focus();
        return;
    }
    else
    {
        MessageBox.show("Opening your application...",
                        "Login Dialog",
                        MessageBox.ICONEXCLAMATION);
        Application.exit();
    }
}

private void CancelPassword(Object source, Event e)
{
    Application.exit();
}
```

该示例的代码非常直观。用户输入完用户名及密码后,就可以单击 btnOK 控件并触发 CheckPassword 方法。CheckPassword 的代码确定用户是否在 txtUsername 控件中输入用户名,如果还没有输入,那么用户将得到警告,并将焦点集中在 txtUsername 控件上。如果已经输入用户名,那么代码将检查用户输入的密码是否与 m_iPassword 中存储的密码一致。

代码使用 ActiveX 控件的 NumFromText 属性(作为 getNumFromText 导入)来获

得输入密码的整数表示法。NumText 控件使您可以获得在控件中输入的数据的字符串及整数值。这是除了 NumText 控件限制条目数目之外,NumText 控件与标准 Edit 控件的主要不同。如果用户输入的密码与 m_iPassword 的值不匹配,那么代码将显示一个消息框,警告用户发生错误,并将焦点集中在 NumText 控件上。如果密码正确,那么代码将显示一个消息框,让您了解要启动的应用程序。因为这两个要求都是满足的,代码以退出应用程序结束。

9.4.6 编译运行

现在编译并运行示例。打开示例后,输入 ntxtPassword 控件,并试着键入一些数字以外的字符。您将会发现,控件不能接受这些字符。输入用户名及当前的密码,看一看消息框是否通知您登录成功。虽然 NumText 控件最初相对简单,但是您将发现将 ActiveX 控件添加到工程中与添加其他更复杂的控件的过程相同。

关于在 WFC 中使用导入的 ActiveX 控件还有一点需要注意。当您与更加复杂的 ActiveX 控件进行交互时,您将很可能不得不在控件中设置值。因为 ActiveX 控件不能使用指定的数据类型,所以您将不得不使用 com.ms.com.Variant 类来指定参数中所需的值。Variant 类的构造函数接受任意类型的 Java 数据来创建对象。虽然它需要进行较多的调用对象的工作,但是并不需要大量的额外代码。

9.5 Java 应用程序与 Java 小程序的混合

在学过 Java 应用程序分类后,我们知道 Java 应用程序分为独立运行的应用程序与小程序,能不能编写单个 Java 代码集,既可作为独立运行的应用程序执行,又可作为小程序在浏览器中执行呢?由于在独立运行的 Java 应用程序和基于 Web 的 Java 小程序中使用 AWT 和 AWT UI 是可能的,所以上面的想法是可以实现的。

9.5.1 应用程序与小程序的混合编程

创建一个可以作为应用程序或小程序的单个 Java 源代码集是可能的。我们知道,独立运行的 Java 应用程序中有 main()方法,它是应用程序的执行点,是 Java 应用程序自动执行的。Java 小程序中的方法 init()、start()和 stop()在 Java

小程序中自动执行。理解了 Java 运行环境处理这两种程序的不同,实现 Java 应用程序与 Java 小程序的混合编程才是可能的。一般来说,Java 应用程序与 Java 小程序的混合编程需要下面的步骤:

- ◆ 在方法 `main()` 中设置标志确保当程序作为单独的应用程序运行时程序不会试图读取 HTML 参数。
- ◆ 对两种参数分别处理。
- ◆ 显式地从方法 `main()` 中调用方法 `init()` 和 `start()`。

应该指出,编写单个 Java 代码集,既可作为独立运行的 Java 应用程序执行,又可作为 Java 小程序在浏览器中执行的有点复杂。这是因为我们要在程序中对两类应用程序分别处理。下面我们通过一个例子来说明这个问题。

9.5.2 Java 应用程序与 Java 小程序例子

考虑第 6 章的例子,它是个 Java 小程序,显示“欢迎来到 Java 的世界!”,根据 Web 网页参数,确定是显示中文还是英文。本节的例子类似,显示“Hello Java”,它的不同点是既可独立运行,也可在浏览器中运行。当 `HelloJava` 作为单个应用程序执行时,它接受命令行参数,根据参数值确定是显示中文还是英文。当作为 Java 小程序执行时,它从 Web 网页读取参数,根据参数值确定是显示中文还是英文。

创建一个空的工程,工程名为 `Hello Java`,添加 `HelloJava` 类,类 `HelloJava` 的代码如下:



```
import java.awt.*;

public class HelloJava extends java.applet.Applet
{

    boolean m_bApplication;
    Label m_label;

    public static void main (String[] argv)
    {
        // TODO: Add initialization code here
        class NewFrame extends Frame
        {
            NewFrame(String title)
```

```
        |
        |    super(title);
        |
    }

    public boolean action(Event evt, Object arg)
    |
        |    if (evt.target instanceof Button)
        |    |
        |        |    String label = (String) arg;
        |        |    if (label.equals("关闭应用程序"))
        |        |        |    System.exit(0);
        |        |        |    return true;
        |        |
        |        |    else
        |        |        |    return false;
        |        |
        |    |
    |
}

Frame f = new JFrame("HelloJava 应用程序");
HelloJava fg = new HelloJava ();
String language = argv[0];
language.toLowerCase();
if (language.equals("english"))
    fg.CreateLabel("Hello Java!");
else
    fg.CreateLabel("你好 Java!");
fg.m_bApplication = true;
fg.init();
fg.start();

f.add("Center", fg);
f.resize(250, 100);
f.show();

|

public void init()
|
    |    if (this.m_bApplication != true)
    |
    |
```

```

        String language = getParameter("mylanguage");
        language.toLowerCase();
        if(language.equals("english"))
            CreateLabel("Hello Java!");
        else
            CreateLabel("你好 Java!");
        this.add(m_label);
    }
    else
    {
        this.add(m_label);
        this.add(new Button("关闭应用程序"));
    }
}

public void CreateLabel(String text)
{
    m_label = new Label(text);
}
}

```

运行此程序,结果如图 9.19 所示,这是默认的情况,显示的是中文。

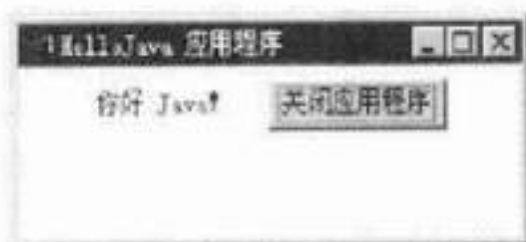


图 9.19 HelloJava 作为独立程序的运行结果

试着将 HelloJava 作为应用程序运行后,可能要将它作为 Java 小程序运行。在工程中添加网页 Page1.htm,使用同样的 HelloJava 可执行码(HelloJava.class)可以做到这一点,网页 Page1.htm 中加载 Java 小程序的代码如下:



```

<APPLET name = HelloJava code = HelloJava.class align = center width =
320 height = 180 >
<PARAM NAME = "mylanguage" VALUE = "chinese" >
</APPLET>

```

在 Internet Explorer 中打开 Page1.htm,如图 9.20 所示。可以在 Page1.htm 中修改参数显示英文。



图 9.20 HelloJava 作为在浏览器中的运行结果

看完运行结果,我们分析一下代码,当 HelloJava 作为应用程序执行时,方法 `main()` 被调用。方法 `main()` 的第一个任务是创建一个 `Frame` 对象。`Frame` 对象将作为应用程序中使用的 AWT UI 控件的容器。由于执行应用程序时 Java 运行环境不会自动实例化类(这一点非常重要),因此 HelloJava 程序中的方法 `main()` 必须显式地创建一个 HelloJava 类的实例。

```
Frame f = new NewFrame("HelloJava 应用程序");
```

此外,对 HelloJava 变量和方法的引用必须指定恰当的实例,否则,编译会出错。方法 `main()` 通过调用方法 `init()` 和 `start()` 模拟小程序。在这种情况下,方法 `start()` 没有被重载。在方法 `main()` 中,设置应用程序布尔型变量(`m_bApplication`)告诉 `init()` 方法,程序正作为应用程序执行。当作为应用程序执行时,我们在程序中判断是否有参数,如果有,根据参数值确定是显示中文还是英文。文本的显示是通过调用 `CreateLabel()` 方法实现的。当作为小程序运行时,在 `init()` 方法中,调用 `getParameter("mylanguage")` 方法来读取 Web 网页中的 Java 小程序的参数,根据参数值确定是显示中文还是英文。

作为独立运行的 Java 应用程序,我们应该有方法关闭它,当 HelloJava 作为应用程序运行时,在 `init()` 方法中,我们添加了【关闭应用程序】按钮。为了响应此按钮事件, `NewFrame` 类扩展了 `Frame` 类,这种扩展是必须的,因为方法 `action()` 必须被重载以便增加对【关闭应用程序】按钮的事件处理。当 HelloJava 作为应用程序执行时这一点才重要。当按钮被点击时, `action()` 方法被调用并用参数 0

调用 `system.exit()` 方法,使应用程序结束。方法 `dispose()` 也可使用,该方法只是释放 `Frame` 对象,而应用程序将继续执行。

在第 6 章,我们介绍过如何使用 Java 小程序参数。对于 Java 应用程序使用参数很简单,像 Dos 命令一样,使用命令行参数即可。在 DOS 窗口下输入如下命令即可。

```
C:\windows\jview c:\JavaBook\samples\Chap09\HelloJava\HelloJava.class
english
```

运行结果如图 9.21 所示,单击【关闭应用程序】按钮,关闭应用程序。在调试环境下使用命令行参数的方法我们将在第 10 章讨论。

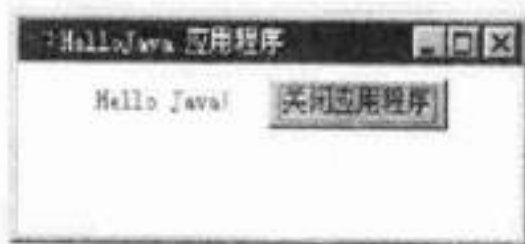


图 9.21 HelloJava 作为独立程序的运行结果(显示英文)

9.5.3 HelloJava.exe

我们在两种环境下运行了 `HelloJava.class`,用 `jview` 执行 `HelloJava.class` 有点不方便,其实我们可把 `HelloJava.class` 编译成可执行文件,这样就可以直接在 Windows 下运行。

打开 `HelloJava` 工程的属性对话框,选择 `Output Format` 选项卡,选中 `Enable Packaging` 复选框,选择 `Package type` 为 `Windows EXE`,`File name` 为 `HelloJava.exe`,选好后,单击 `OK` 按钮,如图 9.22 所示。

重新编译程序,会在 `HelloJava` 目录下生成 `HelloJava.exe` 文件。双击 `HelloJava.exe` 文件,运行结果与图 9.19 相同。为了在运行 `HelloJava.exe` 时使用参数,可创建 `HelloJava.exe` 的“快捷方式 `HelloJava.exe`”文件。右击 `HelloJava.exe`,在快捷菜单中选择【创建快捷方式】,这样就创建了名为“快捷方式 `HelloJava.exe`”的文件,右击这个文件,打开属性页,如图 9.23 所示。

选择【快捷方式】选项卡,在目标位置编辑框中,输入“`c:\JavaBook\samples\Chap09\HelloJava\HelloJava.class english`”。运行快捷方式,结果与图 9.21 相同,显示为英文。当把参数改为 `chinese`,显示为中文。

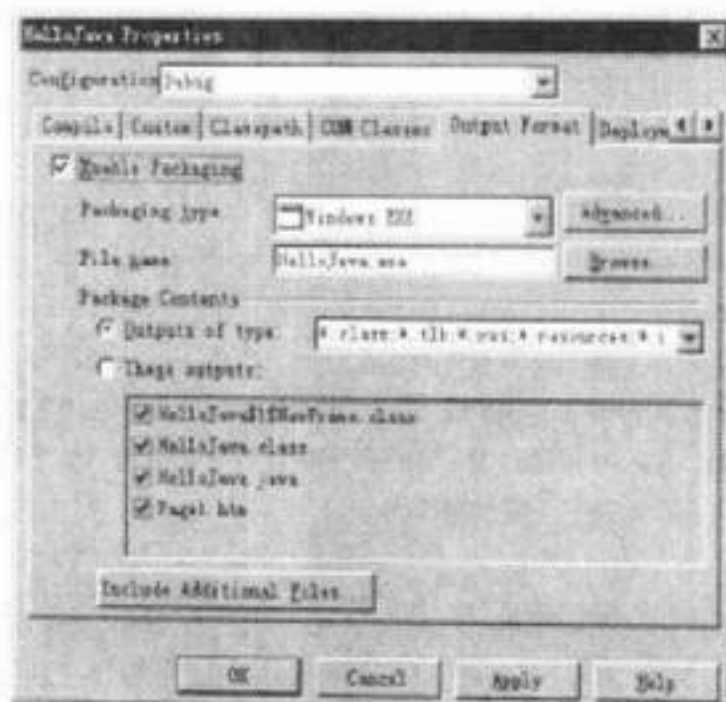


图 9.22 HelloJava 工程输出格式设置

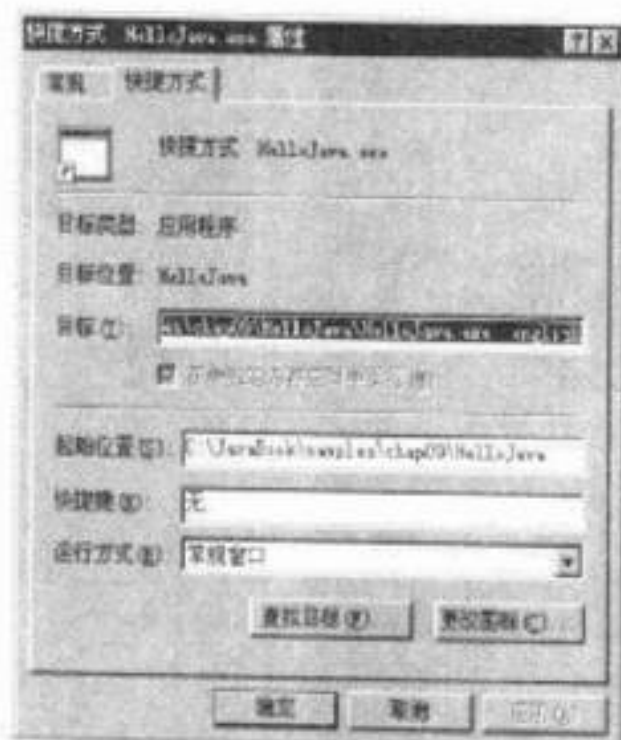


图 9.23 【快捷方式: HelloJava.exe 属性】对话框



更上一层楼

本章我们讲述了如何使用控件,如何在应用程序中增加帮助,如何在应用程序中支持拖放,如何把 ActiveX 控件导入到 WFC 应用程序中。在本章最后一节,我们重点讨论了 Java 应用程序与 Java 小程序运行环境的不同,如何编写既是 Java 应用程序又是 Java 小程序的 Java 代码。作为 Java 应用程序进阶的一部分,有些内容我们没有讲述,比如,编写多线程 Java 应用程序、I/O 操作、安全性等内容。有关 Java 小程序和 Java 应用程序编程部分我们就讲这些内容,下一章我们进入另一个主题——编程调试。

第 10 章 调试

知识要点:

- ◆ Visual J++ 调试器环境
- ◆ 如何调试代码
- ◆ 各种调试工具的综合使用
- ◆ 利用调试窗口进行代码跟踪
- ◆ 如何调试带参数的 Java 控制台程序
- ◆ Java 小程序的调试技巧
- ◆ 调试程序的一般过程
- ◆ 如何调试异常

前面我们没有介绍如何调试程序,一是我们的例子比较简单,二是按着我们的步骤进行,一般不会有问题。如果您学习了前面几章,且自己动手编写示例代码,您一定用到了 Visual J++ 的调试器。对于那些复杂庞大的代码,没有调试器的帮助我们将事倍功半。所以本章我们将介绍 Visual J++ 调试器的一些主要特性,并介绍如何调试处理应用程序中的错误。

调试器是一个功能非常强大的工具,使用该工具可以方便地识别和去除应用程序代码中的逻辑错误。此外,一些由外部软件或硬件环境引起的异常情况的发生,也会影响应用程序的性能,通过添加一些必要的异常处理代码,会避免程序崩溃的事件发生,从而使得程序能够在自己的控制下顺利执行。

经过本章的学习,您不用再怕程序出错了,Visual J++ 调试器的强大功能使得调试程序变得轻松,我们先介绍 Visual J++ 的调试环境。

10.1 调试环境介绍

Visual J++ 调试器基本上具有 Visual C++ 调试器提供的所有特性。由于网络程序复杂,当 Java 小程序涉及到 Web 浏览器、HTML 脚本时,查错是非常困难的。利用调试器,可以在程序运行的过程中观察代码。使用这一功能,就可以使程序在代码段当中逐行运行,在代码段中设置断点,观察变量。在介绍如何调试 Java 程序前,我们先了解一下调试的工具和命令。

10.1.1 调试的工具和窗口

在 Visual J++ 环境中,Debug 菜单如图 10.1 所示。当我们调试程序时,Visual J++ 会弹出 Debug 工具栏。图 10.2 显示了 Debug 工具栏。Debug 菜单项中有各个调试命令的快捷键。表 10.1 列出了主要的调试命令。

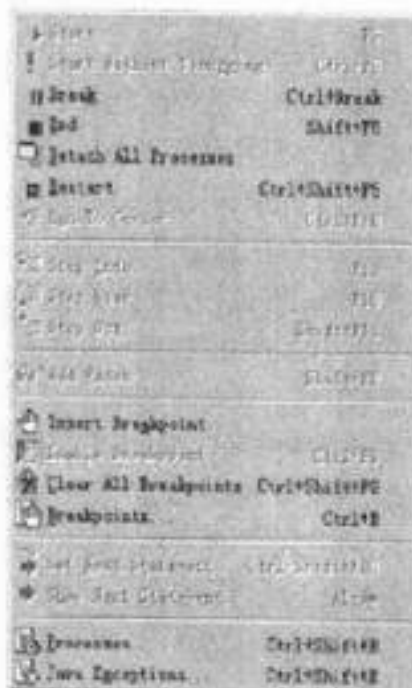


图 10.1 Debug 菜单项

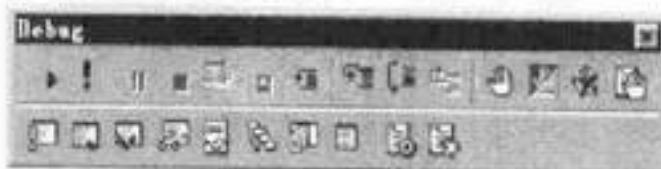


图 10.2 Debug 工具栏

表 10.1 调试命令

命令	快捷键	说明
Start	F5	开始调试运行
Start Without Debugging	Ctrl + F5	非调试下的运行
Break	Ctrl + Break	用于中断正在进行的调试操作
End	Shift + F5	用于终止调试过程,返回到编辑状态
Restart	Ctrl + Shift + F5	表示重新对程序进行调试
Run to Cursor	Ctrl + F10	用于调试程序执行直到插入点处
Step Into	F11	用于逐步调试程序,而且遇到程序中调用的函数时,进入函数内部执行
Step Over	F10	逐步调试程序,但是当遇到调用函数时,并不进入函数内部执行
Step Out	Shift + F11	在进行调试程序时,从正在执行的某个嵌套结构的内部跳到该结构的外部,常用于知道调用函数中不存在错误的情况
Add Watch	Shift + F9	用于快速查看表达式、变量等的值
Insert Breakpoint		插入断点
Enable Breakpoint	Ctrl + F9	激活断点
Clear All Breakpoints		清除所有断点
Breakpoints		查看所有的断点
Java Exception	Ctrl + Shift + E	显示 Java Exception 对话框

除了 Debug 命令外,还有用于查看调试信息的各种窗口,在 Visual J++ 环境下,打开 View 菜单,单击 Debug Windows 命令,可以看到有多种不同的调试窗口,如图 10.3 所示。

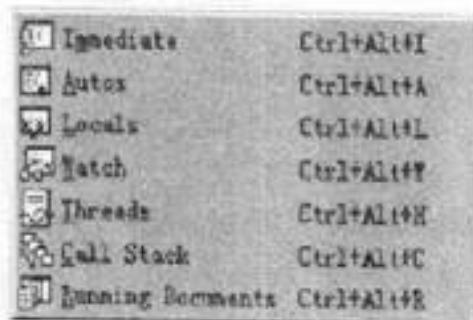


图 10.3 查看调试信息的各种窗口

有关调试窗口的使用,我们将结合例子介绍它们的使用方法。下面介绍调试前的准备工作。

10.1.2 调试前的准备

一般情况下,我们在 Debug 菜单下运行程序,会自动进入调试状态。此时调试器使用默认设置。在调试程序前,为调试规定适当的设置非常重要。

本章我们用第 6 章的 ex06d 工程为例,来说明如何使用 Visual J++ 调试器的功能。调试之前,最好备份 ex06d 工程,打开此项目,确定调试程序后,打开 Build 菜单,单击 Build Configuration 命令,选中 Debug 项,表示工程在调试模式下。此时打开工程属性对话框,当前显示的是 Debug 的一些设置,如图 10.4 所示,单击 OK 按钮,开始调试程序。



图 10.4 ex06d 工程的 Debug 的配置

如果程序有语法错误,要先排除错误再调试。打开 Build 菜单,选择 Build 命令,Visual J++ 编译工程,在任务列表窗口中显示编译信息。

例如,把 ex06d 工程中的 Class1.java 代码中的 paint(Graphics p1)函数的 language.toLowerCase()改为 language.qtoLowercase()后,这时就无法通过编译,如图 10.5 所示。红色的惊叹号表示有语法错误,在 Description 列中有错误的描述信息和错误代号。根据错误信息提示纠正错误。错误代号是错误信息说明的编号,通过 MSDN 可得到更详细的错误说明。

如果没有编译错误,我们就可以调试程序了。

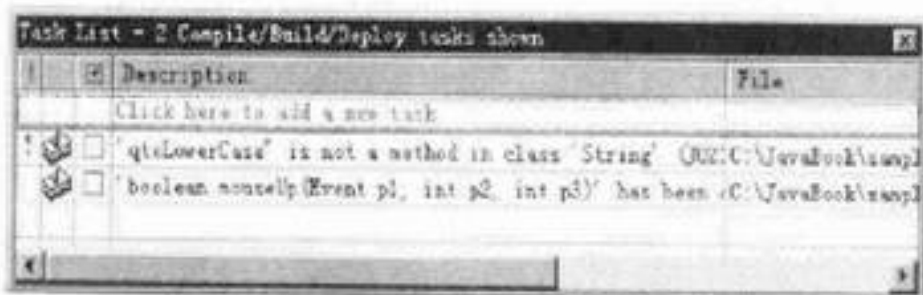


图 10.5 任务列表窗口中显示的编译信息

10.2 调试代码

到目前为止,我们已经运行了应用程序,但是,这种运行是通过单击 Start Without Debugging 而实现的。现在,我们使用 Debug 工具栏中的 Start 按钮,来实现应用程序的运行,这样,就可以使应用程序在调试模式下运行,也就是说,在这种运行模式之下,调试器的所有功能都可以使用了。

10.2.1 断点

为了在程序的运行过程当中检查代码,就要在运行中暂停程序。为了达到这个目的,就要设置断点。断点是程序调试人员在代码中的某行所添加的标记,当程序运行到该处时,编译器会将程序暂停在此处。设置断点的方法有以下 4 种:

- ◆ 选择要设置断点的代码行,按下 Insert Breakpoint(插入断点)按钮(该按钮上绘有一个小手标记)。
- ◆ 选择要设置断点的代码行,然后在 Debug 菜单中选择 Insert Breakpoint 命令。
- ◆ 右击要设定断点的行,在快捷菜单中选择 Insert Breakpoint 命令。
- ◆ 单击代码窗口边缘的灰色区域。

设置了断点以后,当应用程序开始运行时,程序会一直运行到设置了断点的代码处,然后暂停。在代码窗口中,设置了断点的代码行在窗口左侧边缘会有一个停止的标记,您可以很容易地判断断点即程序运行到的位置,如图 10.6 所示。



图 10.6 代码窗口左侧边缘的停止标记示意断点所在的位置

若要清除或移动断点,可用如下方法:

- ◆ 将鼠标指向设置了断点的行,单击鼠标右键,弹出快捷菜单,如图 10.7 所示,选择 Remove Breakpoint 命令。
- ◆ 在 Debug 工具栏中按下 Remove Breakpoint 按钮。
- ◆ 用鼠标直接单击断点标记,即可清除断点。

如果想清除所有断点,可打开 Debug 菜单,使用 Clear All Breakpoints 按钮,可以清除所有的断点。



图 10.7 与断点相关的快捷菜单

除此之外, Visual J++ 还提供了完整保留断点而在调试运行时忽略断点的功能,这也被称为禁用断点。在任何具有 Remove Breakpoint 选项的菜单或工具栏中,也都具有 Disable Breakpoint 选项。禁用断点的标记在代码窗口左侧为空白的停止标记。

如果想要查看所有的断点,可以使用 Breakpoints 按钮。此时,屏幕上就会显示如图 10.8 所示的对话框。

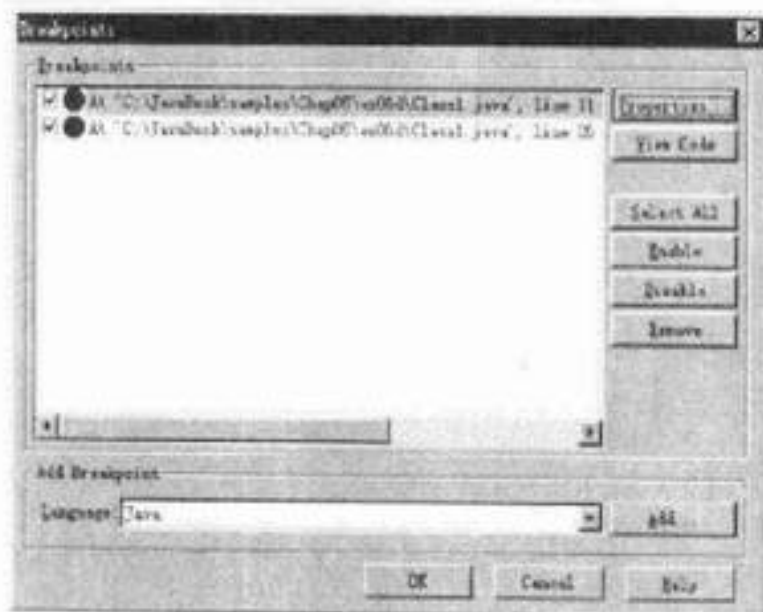


图 10.8 断点窗口

使用该窗口,可以看到工程文件中的所有断点。这个窗口中包括断点所在的源代码文件名、所在行以及方法名称和方法所在的行。如果选中了某个断点的复选框,该断点就会被激活(也就是说,程序运行到该断点处,调试器会停止程序)。如果断点的复选框没有被选中,那么,该断点在调试过程中将不起作用。另外,也可以在此窗口中选中要删除的断点,然后使用 Remove 按钮,将断点彻底清除。在该窗口还可以添加断点,但是,在源代码窗口中添加断点更为容易。

在图 10.7 和图 10.8 中的窗口中,我们可以查看断点的属性。断点属性对话框如图 10.9 所示。

在断点的属性对话框中,选择 Hit count 复选框,可计数断点的数目。默认是不计数的,图 10.9 显示的是选中计数,“if(language.equals(“english”))”语句执行两次的结果。此功能在调试循环、函数调用时很有用。

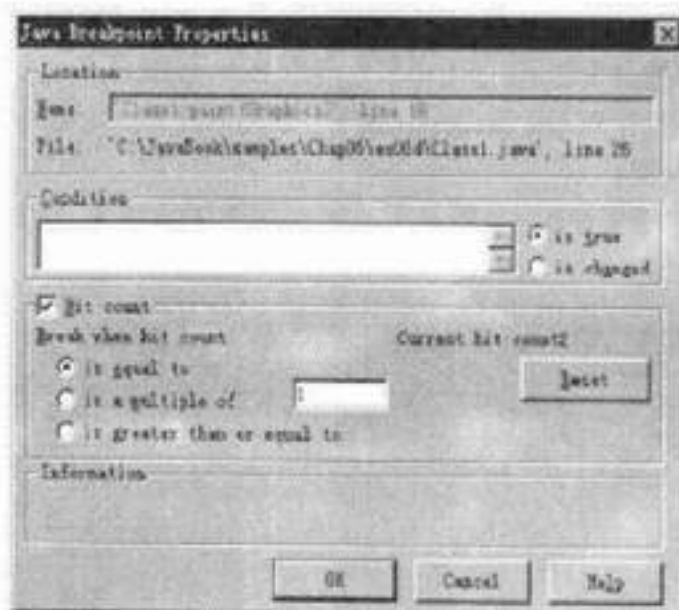


图 10.9 断点的属性

10.2.2 执行到光标处

与断点类似,执行到光标处(Run to Cursor)命令使程序执行到光标处停止,有时用它很方便,比如查看某个条件是否为真,变量的值是多少等。

断点和执行到光标处都需要单步运行命令配合使用才能发挥更大的作用,可以查看程序运行时的动态信息。

10.2.3 在源代码中单步运行

在源代码中添加了断点之后,就可以按下 Start 按钮,开始运行应用程序;或者打开 Debug 菜单,选择 Run to Cursor 命令。在此之后,屏幕上就会出现如图 10.10 所示的 Watch 窗口,与此同时,调试器开始运行代码。可以及时查看 Watch 窗口。

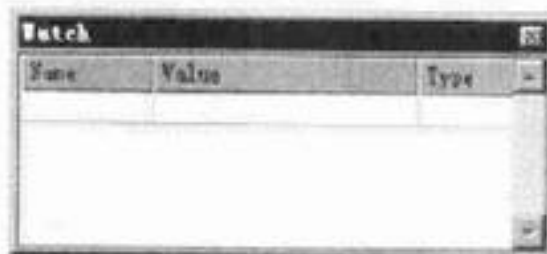


图 10.10 Watch 窗口



注意 对于 Visual J++ 环境,在调试模式和设计模式下,屏幕上所显示的窗口往往是不同的。比如说,当处于程序设计工作模式下,在屏幕上显示 Watch 窗口就没有必要了。所以,当屏幕上窗口的状态发生变化时,所熟悉的窗口没有出现,也不必大惊小怪。完全可以在 View 菜单当中选择合适的选项,以显示希望使用的窗口。但是,无论怎样都要确保所选择的窗口是与工作模式相对应的。

应用程序在停止之前究竟运行到什么地方,这完全取决于第一个断点所设定的位置。例如:如果将断点设置在一个按钮控件的事件处理程序上,那么,就必须按下这个按钮,使应用程序停止在该代码行。当程序运行到设置了断点的代码行,代码窗口左侧边缘对应该行的位置上,就会出现一个箭头作为标记,表示程序运行到此处,如图 10.11 所示。



图 10.11 调试运行模式下的代码窗口,在某断点处停止

现在,可以查看应用程序的当前状态,或者通过该代码行向下运行。在 Debug 工具栏中,所有按钮都具有专门的功能对应于不同的调试方案。

若要使调试运行工作往下进行,在 Debug 工具栏中,可以选择几个按钮,如果当前行所包含的方法中有需要访问的源代码,可以选择以下几项:

- ◆ **Step Into**。选择此按钮,调试器将直接运行到该方法处,并且将当前行设定为该方法的第一行。
- ◆ **Step Over**。选择此按钮,调试器将不显示代码的任何信息,而直接运行该方法的代码。
- ◆ **Step Out**。选择该选项时,调试器将运行当前方法的其他有关代码,直至方

法调用结束。

如果当前行不是一个方法,或者该行中没有所要访问的源代码,那么 Step Into 和 Step Over 按钮的作用就是相同的。另外,要注意的是,即便选择 Step Out,断点仍然是有效的。如果选择跳过一个方法,断点将在该方法结束前生效,程序仍然要在断点处停止。断点的语句运行完后,如果使用单步调试,只要程序再次执行到此语句,断点仍然是有效的。



注意 当逐步调试运行程序代码时,总会有这样的时候,应用程序不显示任何内容。此时不必担心,对于您感兴趣的值,调试器会随着程序的执行而及时地显示出来。您可以使用应用程序和 Visual J++ 工程文件环境之间的任务栏上的前进和后退按钮反复查看,直到对调试过程有了全面的了解为止。

10.2.4 Watch 窗口

Watch 窗口,如图 10.12,可以用来监视表达式和变量,当使用调试器开始运行一个应用程序以后,屏幕上就会出现 Watch 窗口,其中有许多空白区域。这个窗口中的区域正是为您所感兴趣的变量而准备的。如果想将某一变量移动到查看窗口中,首先右击该变量,在快捷菜单中选择 Add Watch 命令。这样,系统就会将所选择的变量放到 Watch 窗口中标有 Name 的栏中,与此同时,窗口中还会显示出,变量的值以及变量类型的简要说明。您也可以通过从代码窗口中向四个栏中的任何一个拖放来增加表达式和变量。如果程序继续运行到某行代码,使得变量的值发生变化时,变量值就会变成红色。当程序运行到下一行代码时,变量值又会变为原来的黑色。例如,将程序中的 name 变量添加到 Watch 窗口中。图 10.12 显示了变量在 Watch 窗口中的情形。

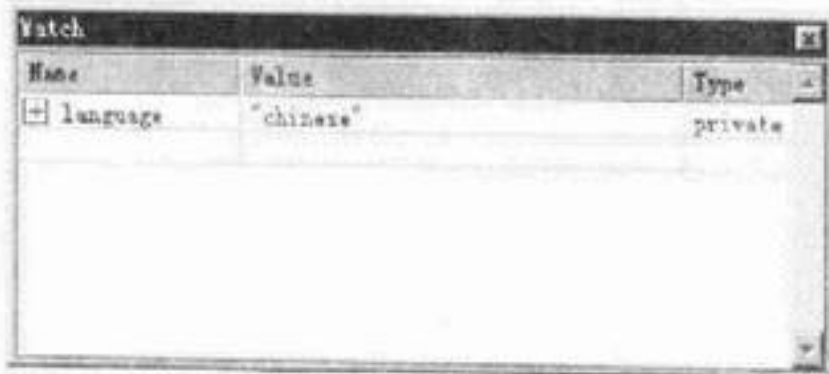


图 10.12 Watch 窗口及窗口中的变量

如果此时右击 Watch 窗口中的变量,就会在快捷菜单中得到几个选项,可以复制 Watch 窗口中的内容,也可以移动该变量在 Watch 窗口中的位置,以及继

续往下进行。

10.2.5 Immediate 窗口

在程序调试的过程中,往往会碰到这样的情况,就是在对程序进行单步调试时,需要及时知道某个变量的值。下面,将介绍 Immediate 窗口是如何实现这一功能的。

当我们跟踪某个变量发现它的值不对时,我们往往停止调试,排除错误后再调试,利用 Immediate 窗口我们不必遇到错误就停止调试,因为它可以改变变量的值。

Immediate 窗口可以通过向变量中存储数值的办法,来实现在运行的过程中改变变量的值。如果在 Immediate 窗口中输入一个变量名,并按下 Enter 键,窗口中就会显示出变量的值。此时若要改变变量的值,在 Immediate 窗口中输入赋值表达式即可。还是以 language 变量为例,图 10.13 显示了这个过程。在窗口中输入变量 language,并按下 Enter 键,在此之后,屏幕上显示出该变量的值“Chinese”。

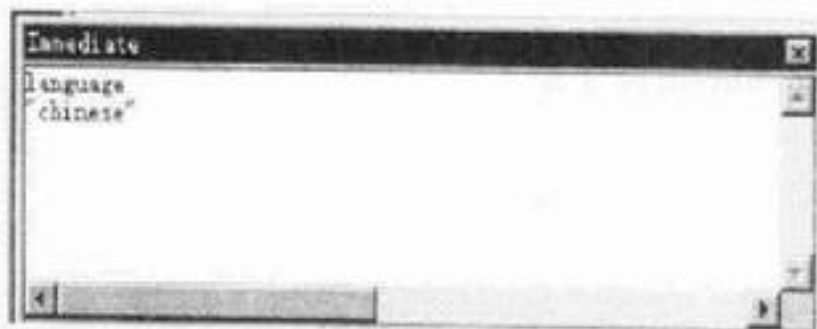


图 10.13 使用 Immediate 窗口

10.3 调试器的其他窗口

Visual J++ 调试器除了 Watch 窗口、Immediate 窗口,还有 Output 窗口用于调试 Java 控制台程序的输出,Autos 和 Locals 变量窗口。下面我们通过例子说明它们的作用。

10.3.1 调试例子

为了调试 Java 控制台程序,我们用 Visual J++ 向导创建一个简单的控制台应用程序 ex10a,自动生成的代码如下:



```

/*
 * This class can take a variable number of parameters on the command
 * line. Program execution begins with the main() method. The class
 * constructor is not invoked unless an object of type 'Class1'
 * created in the main() method.
 */
public class Class1
{
    /*
     * The main entry point for the application.
     *
     * @param args Array of parameters passed to the application
     * via the command line.
     */
    public static void main (String[] args)
    {
        // TODO: Add initialization code here
    }
}

```

我们用这个例子说明如何利用 Output 窗口、Autos 和 Locals 变量窗口调试控制台程序。

10.3.2 Output 窗口

当 Java 在命令行模式的操作系统(例如,MS-DOS 或 UNIX 操作系统)下运行时,程序的输出将写到控制台。另外,写入控制台的数据是基于字符的,也就是说,没有窗口支持。Visual J++ 是在 Output 窗口中显示输出的。为了说明这一点,我们修改上面的 main()函数为:



```
public static void main (String[] args)
{
    System.out.println("Hello World");
}
```

该程序在 Visual J++ 中,“Hello World”将显示在 Output 窗口中,如图 10.14 所示。



图 10.14 Output 窗口

在调试过程中,可以方便地使用 Visual J++ 的这一功能。在跟踪程序的运行进程时,可以通过在程序中任何位置放置如下代码行:

```
System.out.println("行到某某方法中");
```

从而将所需的信息写到 Output 窗口。在程序中添加了这样的代码行以后,当程序运行到此处时,系统就会将括号中的字符串写入 Output 窗口,同时又不影响应用程序的其他部分。打开 View 菜单,在 Other Windows 子菜单中选择 Output 命令,就可以打开 Output 窗口。如果在实际操作时,Output 窗口没有显示出上面所说的内容,此时,应该检查 Output 窗口的顶部下拉框中是否有 Debug(而不是 Solution Builder)。

当然,当经过调试运行确保自己的程序满足要求时,就一定希望在运行时不要对系统的 System.out.println 库文件进行任何调用。

控制台程序可能有参数,如何在调试中使用参数我们可以在工程的属性对话框中设置命令行参数,修改 main() 函数如下:



```
public static void main (String[] args)
{
    // TODO: Add initialization code here
}
```

```
String instr = new String("");  
instr = args[0];  
instr += " world";  
System.out.println(instr);  
}
```

我们想传递参数为“Hello”，输出结果仍为“Hello world”，当然传递不同参数时，输出结果不同，我们只是说明如何在调试中使用参数。打开 ex10a 工程的属性对话框，选择 Launch 选项卡，选择 Custom 单选按钮，设置 Program 为：“JView.exe”设置 Arguments 为：“/p /cp:p "<JAVAPACKAGES>" Class1 Hello”。

如图 10.15 所示：



图 10.15 Launch 选项卡的设置

这样，就会把参数传递给控制台应用程序。此时调试运行，Output 窗口输出结果同图 10.16 所示。

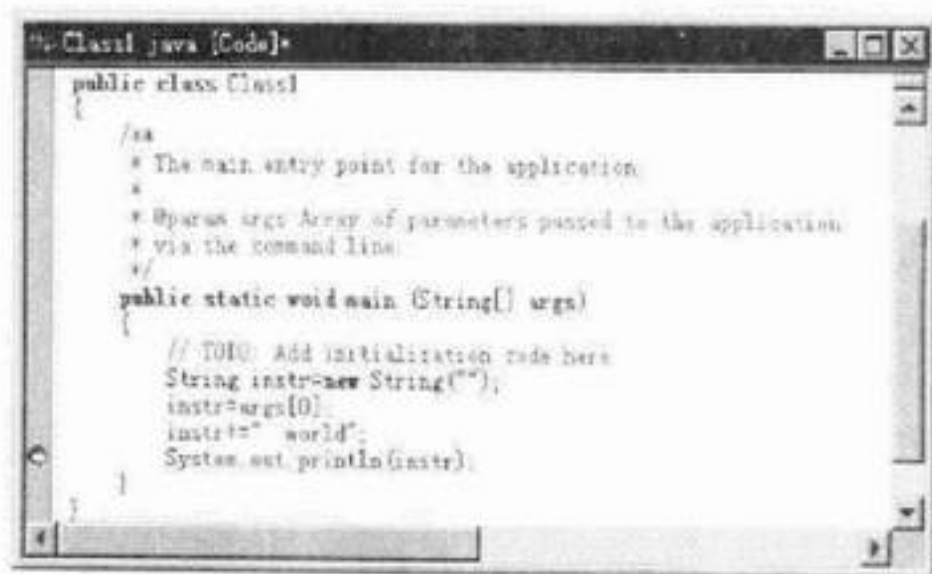


图 10.16 为查看 Autos 窗口而设置的断点

10.3.3 Autos 窗口

Autos 窗口显示当前方法的作用域中的所用变量,它不像 Watch 窗口、Immediate 窗口那样需要输入变量名,它将自动显示变量名、值和类型。仍以 ex10a 工程为例,设置断点如图 10.16 所示。然后在调试环境下运行程序,查看 Autos 窗口,如果 Autos 窗口没有打开,先打开 View 菜单,在 Other Windows 子菜单中选择 Autos,即打开 Autos 窗口。运行到图 10.16 所示断点时,Autos 窗口如图 10.17 所示,有变量名、值和类型。单击变量名前的“+”号,可查看变量的详细情况。图 10.18 显示的是字符串变量 instr 的详细情况。

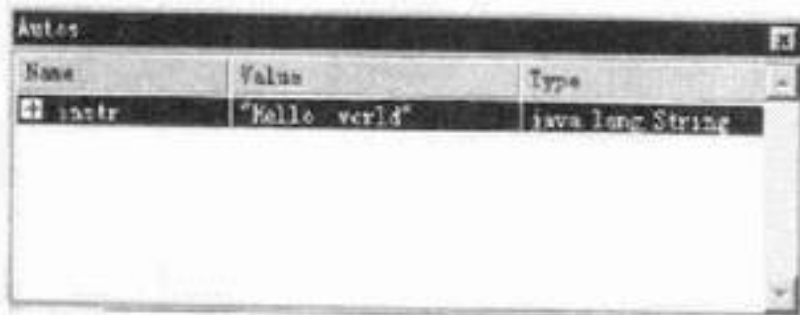


图 10.17 Autos 窗口

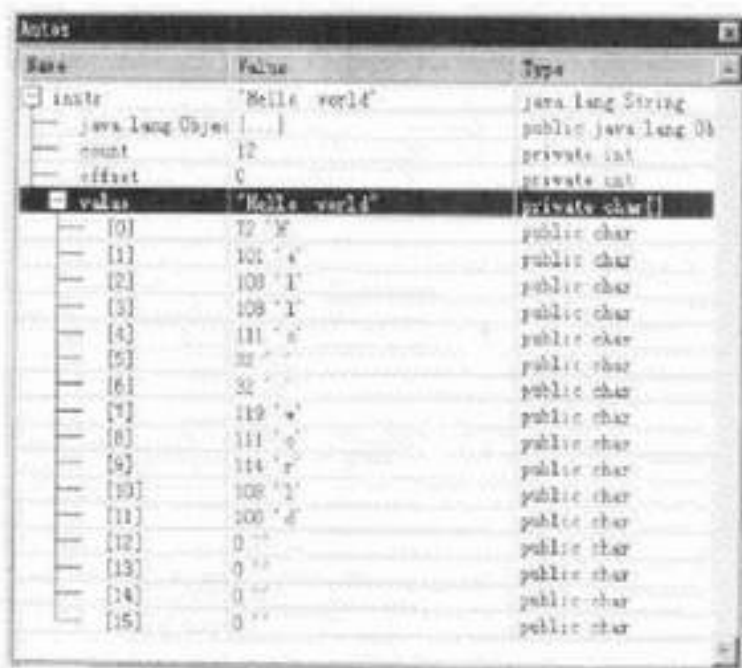


图 10.18 字符串变量 instr 的详细情况

10.3.4 Locals 窗口

Locals 窗口显示当前方法的局部变量,它与 Autos 窗口类似,还用图 10.16 的断点,然后在调试环境下运行程序,查看 Locals 窗口,如果 Locals 窗口没有打开,先打开 View 菜单,在 Other Windows 子菜单中选择 Locals 命令。运行到图 10.16 所示断点时,Locals 窗口如图 10.19 所示,同 Autos 窗口类似,Locals 窗口有变量名、值和类型,在 Locals 窗口上面列表框种类除了局部变量。单击图 10.19 中变量名前的“+”号,可查看变量的详细情况。图 10.20 显示的是字符串变量 args 的详细情况。

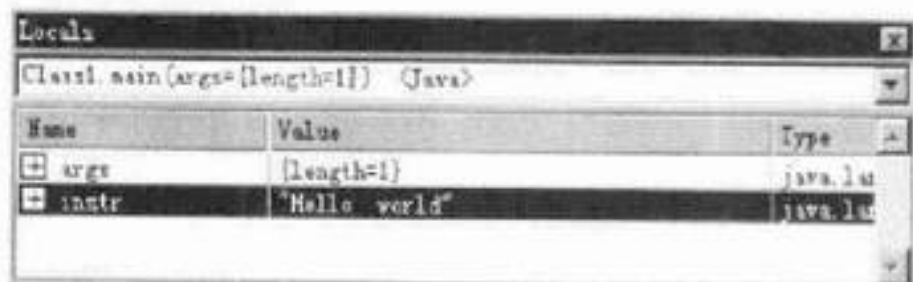


图 10.19 Locals 窗口

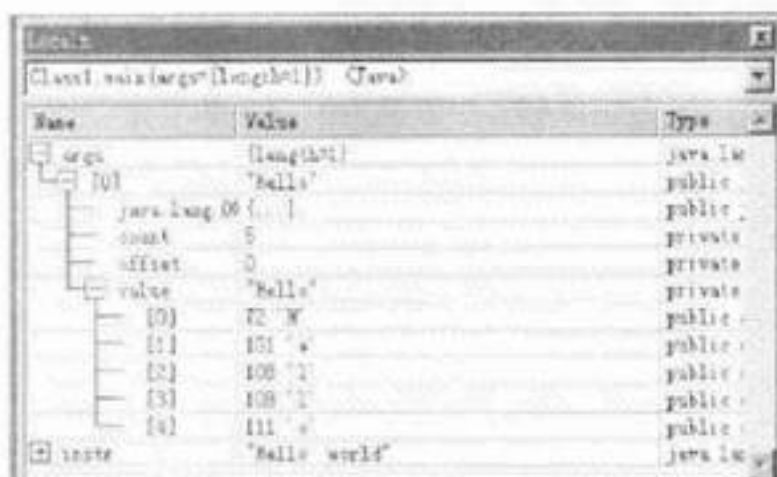


图 10.20 locals 窗口变量 args 的详细情况

10.3.5 Call Stack 窗口

调用堆栈窗口,如图 10.21 所示,显示到当前执行点时,发生在调试期间的
所有方法的调用层次。

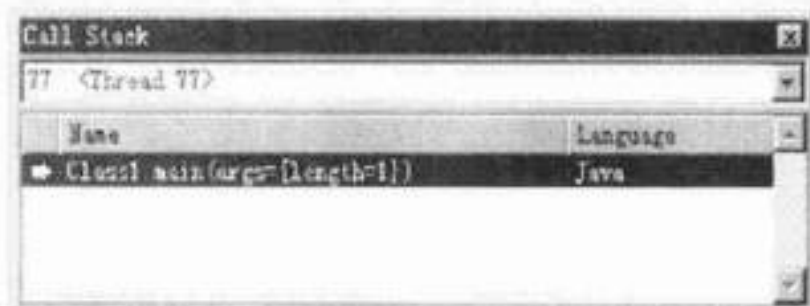


图 10.21 调用堆栈窗口



注意 10.2 和 10.3 节中介绍的调试窗口对所有的 Java 应用程序都是适用的。

10.4 Java 小程序的调试技巧

前面介绍的调试方法对 Java 小程序也是适用的。本节我们介绍 Java 小程序
与 Web 网页有交互时,如何调试 Java 小程序。当 Java 小程序有参数时怎样调试

呢?当然可以先在程序中把参数写定,调试通过后,再设置参数。Visual J++ 调试器提供了综合调试功能,Java 小程序和 Web 浏览器直接相关,直接调试。

10.4.1 ex06d 例子再调试

为了说明问题,我们看看调试 ex06d 例子的情况,ex06d 例子是从 Web 网页中读取参数,根据参数值,确定输出是中文还是英文。默认情况下是中文。设置断点如图 10.22 所示。

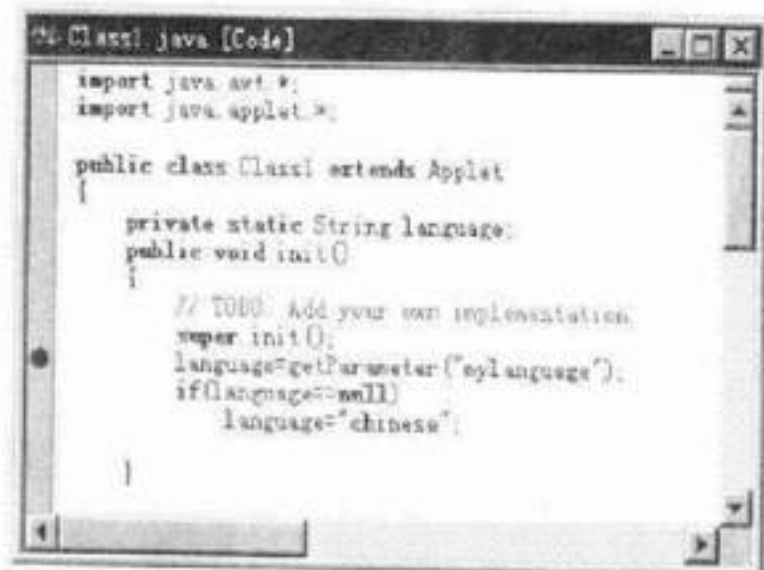


图 10.22 ex06d 例子的断点

用 Watch 窗口观察,单步执行完语句“language = getParameter(“mylanguage”)”;后,变量 language 的值没有变化,一直为 null。这说明参数没传递进来。为传递参数,需要设置 Java 小程序所在工程的属性。

10.4.2 设置 Java 小程序所在工程的属性

出现上面情况的原因,在 6.6 节中已说明,在 Applet view 看不到传递参数的结果。因此用 Applet view 不能传参数给 Java 小程序,为此我们使用 Internet Explorer 浏览器,用 Web 网页直接传参数给 Java 小程序。还是以 ex06d 为例,打开 ex06d 工程,为了使用 Internet Explorer 浏览器,打开工程的属性对话框,选择 Launch 选项卡,选择 Custom 单选钮,设置 Program 为 Internet Explorer 浏览器所在路径为:

C:\Program Files\Plus! \Microsoft Internet \EXPLORE.EXE

设置 Arguments 为加载 Java 小程序的 Web 网页所在路径为:

file:///C:/JavaBook/samples/Chap06/ex06d/Page1.htm

设置完后如图 10.23 所示。



图 10.23 ex06d 工程 Debug 属性页的设置

Launch 选项卡按上述要求设置完后,我们就可以用调试器查看 Page1.htm 传给 Java 小程序的参数是什么。

10.4.3 跟踪 Java 小程序的参数

使用图 10.6 的断点,然后在调试环境下运行程序,运行到断点时,Watch 窗口如图 10.24 所示。

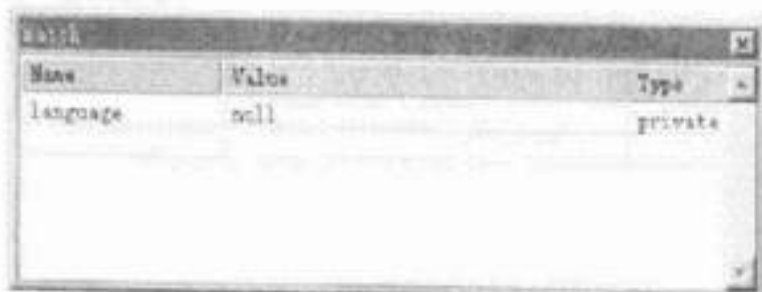


图 10.24 读取参数前变量 language 的值

按 F10 键单步执行一次, Watch 窗口如图 10.25 所示。

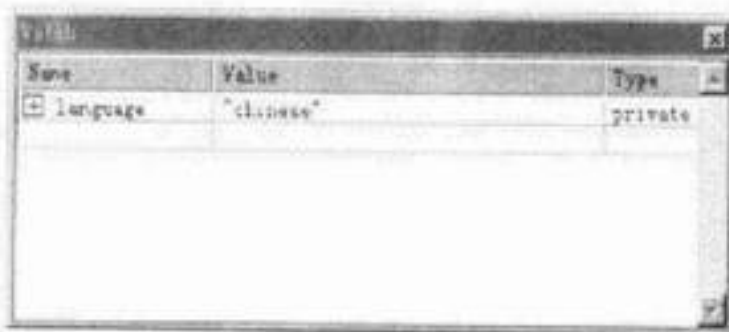


图 10.25 读取参数前变量 language 的值

由于在 Page1.htm 中,我们设置了 Java 小程序的参数 mylanguage 为"chinese",因此执行完语句 language = getParameter("mylanguage")后,language 值为"chinese"。

10.5 调试过程

调试排错是排除程序运行时的错误,这和编译错误不同,也就是我们通常所说的应用程序错误是指由于某些可控参数或条件的影响,使某个函数或者进程不能顺利执行指定的任务,或执行的结果与我们的期望不同。与编译错误相比它更难排除。

整个调试过程包括以下 4 个主要步骤:编译、运行、调试、修改。而且 4 个步骤是个重复循环的过程,经过多次编译、运行、调试、修改的循环,应用程序才能正确、稳定运行。

10.5.1 编译、运行

利用 Visual J++ 编译器,排除编译错误,一般来讲,这一步不会很难,往往是输入错误和语法、语义错误。利用 Visual J++ 编译器的提示信息,比较容易找出出错原因。

编译通过后,运行程序,如果是所期望的结果,调试过程结束,这当然是理想情况。一般来讲,运行时总会有些问题,尤其是代码比较多,算法比较复杂时,运行出错不可避免,这时就要用 Visual J++ 调试器来调试程序。

10.5.2 调试、修改

运行时出错后,首先判断错误可能的出处,然后利用 Visual J++ 调试器,可以在程序运行的过程中观察代码,可以使程序在代码段中逐行运行;在代码段中设置断点,观察变量,步步跟踪,直到找到出错原因。然后修改代码,再编译、运行、调试。

在调试运行的过程中,一旦检查并通过了可能发生错误的代码,就可以准备继续运行程序,直到下一个断点,此时应该再一次单击 Start 按钮(这时 Start 按钮的提示为 Continue)。如果在调试时想要程序运行到某个位置停止,但是又不想在那里设置断点的话,可以使用运行至光标处(Run To Cursor)按钮。单击该按钮,程序就会由当前位置一直运行到光标所指定的代码行(如果此时尚未开始调试,那系统就自然从第一行开始运行到所选定的行)。但要注意的是,如果在程序运行的当前行与光标所选定的代码行之间存在断点,并且这些断点处在激活状态时,调试器会使程序首先在这些断点处停止。如果此时想忽略某个断点,要么将其删除,要么就将其禁用。

如果想停止应用程序,可以选择 Debug 菜单中的 End 命令。

当我们熟练掌握了调试方法后,可多设断点,调试一次,排除多个错误。这样会缩短调试时间。

10.6 调试异常处理

“异常”指的是程序运行时出现的非正常情况。在用传统的语言编程时,程序员只能通过函数的返回值来发出错误信息,这容易导致很多错误。因为在很多情况下需要知道错误产生的内部细节。Java 有异常类来处理异常,从而避免系统的崩溃。

异常处理方法,通常的做法是将错误处理代码内置在函数中,以确保程序的顺利执行,一旦发生严重错误,错误处理代码就会终止应用程序的执行,从而避免系统的崩溃。

10.6.1 Java Exceptions 对话框

调试程序时对异常的处理由 Java Exceptions 对话框进行设置。打开 Debug 菜单,选择 Java Exceptions 命令,弹出 Java Exceptions 对话框,如图 10.26 所示。

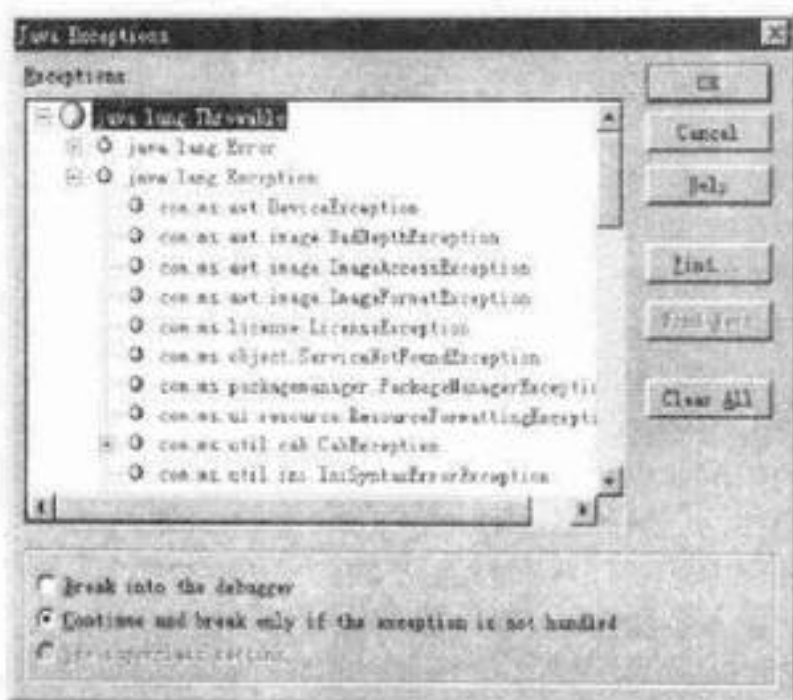


图 10.26 Java Exceptions 对话框

该对话框用来指定在调试期间程序如何对指定的 Java 异常情况进行处理。Exceptions 部分是异常列表,异常列表是个树状结构,按层次结构包括全部的 Java 异常。默认情况下,如果异常发生,有处理例程,异常不会停止程序的执行。

对话框下面的 3 个单选按钮为异常设置,当从异常列表中选中了某个异常类型后,异常设置决定了如何处理这个异常。

单击 Find 按钮,弹出文本输入框,如图 10.27 所示,在编辑框中输入异常名,单击 OK 按钮,在异常列表中查找相应的异常,并从异常列表中显示此异常。查到后,在异常设置选项中进行设置。下面我们讲述异常处理方式的设置。

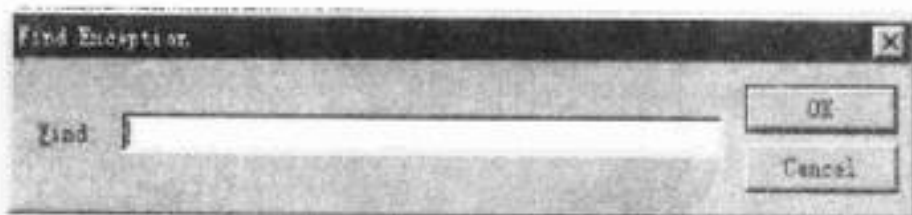


图 10.27 异常查找对话框

10.6.2 异常设置(Exception Settings)

当我们对某个异常进行处理时,可在异常列表中选中此类异常,通过异常设

置选项设置对此类异常发生时的处理方式。异常设置有 3 个选项：

- ◆ Break into debugger
- ◆ Continue and break only if the exception is not handled
- ◆ Use setting from superclass

中断调试器(Break into debugger)

当从异常列表中选中的异常类型发生时,会显示一个消息框,提示异常发生。如图 10.28 所示,可中断程序的执行,也可忽略继续执行。

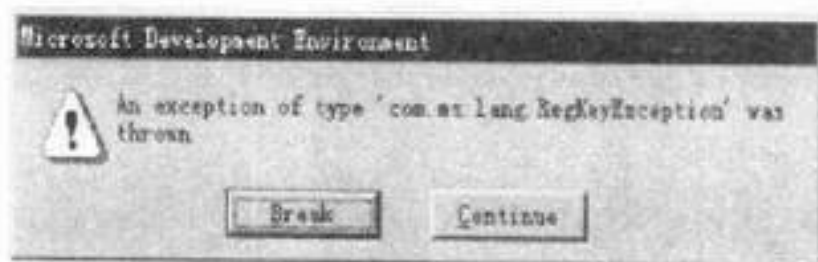


图 10.28 异常发生时的消息框

Continue and break only if the exception is not handled

当从异常列表中选中的异常类型发生时,在没有异常处理代码时,调试器将终止程序的执行。当发生异常情况时调试器将向 Visual J++ 窗口写一条消息。

使用父类设置(Use setting from superclass)

选用此设置,当从异常列表中选中的异常类型发生时,按其父类异常处理设置进行处理。

如果对异常处理设置有误,可使用 Clear All 命令,清除所有的异常处理设置,使用系统默认设置。



更上一层楼

在本章中,我们主要介绍 Visual J++ 调试器的一些主要特性,并介绍如何调试处理应用程序中的错误。我们通过两个例子详细介绍了如何使用调试工具和调试窗口来调试 Java 程序,同时我们也介绍了调试程序的一般过程和如何调试异常。Visual J++ 调试器是一个功能非常强大的工具,使用该工具可以方便地

识别和纠正应用程序代码中的逻辑错误。本章中我们讲述的是基本的调试方法。对于一些复杂程序的调试我们没有展开深入的讨论,比如多线程的调试、Web 网页中有 ActiveX 控件、Vbscript 或 Jscript 脚本、Java 小应用程序时如何调试。正如本章开头所提到的,利用 Visual J++ 调试独立运行的 Java 程序比较容易,而调试 Java 小程序有时则要困难一些,这是因为 Java 小程序的运行环境比较复杂,涉及的技术面比较广。读者有兴趣可查阅相关资料深入了解 Visual J++ 测试器。

第 11 章 数据库编程

知识要点:

- ◆ 什么是 ADO? 为什么用 ADO?
- ◆ 通过数据控件访问数据库
- ◆ 使用数据控件
- ◆ 通过代码直接访问数据库
- ◆ 使用数据库应用程序向导——Data Form Wizard

本章我们将学习编写数据库应用程序和有关数据库编程的常见技术。首先介绍 Visual J++ 提供的数据库对象——ADO (ActiveX Data Objects), 并使用 ADO 和控件将数据库功能添加到 WFC 应用程序中。此外还将学习 ADO 的数据绑定模式, 并将用 ADO 创建简单但实用的应用程序, 学习如何使用 DataGrid 控件来提供允许进行修改的记录集的多记录视图, 以及如何通过代码直接访问数据库。最后介绍创建数据库应用程序的向导——Data Form Wizard。



光盘 参阅本书配套光盘中的【数据库】部分可交互学习与本章相关的知识。

11.1 为什么用 ADO

11.1.1 早期的数据库

访问数据是所有应用程序的基本功能。任何应用程序都需要和数据打交道,从零售点的操作到总公司的库存系统,应用程序都必须能够方便灵活地访问数据。最流行、最实用的应用程序是使用数据库来进行数据交互。

数据库可以存储在本地硬盘上,也可以存储在较大的数据库服务器上。多年来,使用数据库意味着为您使用的数据库平台编写特定的代码。如果您编写的应用程序需要从各种源读取数据,那么这种访问数据库的方法就会产生问题。为了缓解这种局面,诸如 Visual Basic 和 Visual C++ 的编程语言环境提供了访问数据库的对象。这些早期的数据库对象有些是被设计用来处理本地数据库,如 Microsoft Access 等。有些则被设计用来处理远程数据库,例如 Microsoft SQL Server 等。随着数据库应用程序的发展并集成更多的商业用途,一些公司需要能够很容易地扩展它们来适应更大更有效的平台。因此,对访问各种类型及规模的数据库的对象的需求也越来越迫切。

11.1.2 为什么选择 ADO

ADO 是 ActiveX Data Objects 的缩写,即 ActiveX 数据对象。ADO 是一组允许连接到并使用任意类型的数据库但不管它的位置的数据库对象。ADO 允许使用任何链接和嵌入数据库(OLE DB)数据提供者的对象。OLE DB 数据提供者使您可以链接到多个数据库。使用数据提供者,可以访问电子邮件和文件系统,以及简单的图形对象和文本文件。使用 ADO,应用程序可以访问几乎所有的数据。ADO 还被设计用来将本地和远程数据对象集中到一个可以用来连接到任意数据源的集合中。除了数据对象之外,ADO 还包括一些控件,它们使您可以方便地访问数据库字段并将它们绑定到应用程序中的控件中。



注意 ADO 被设计用来集中 Microsoft 的“数据访问对象”(DAO)及“远程数据对象”(Remote Data Objects, RDO)数据库编程模式,来创建一个语言独立的数据库编程模式。因为 ADO 是一个相对较新的内容,所以它还没有提供 100% 的 DAO 和 RDO。最明显的一点是不能使用 DAO 的数据定义功能来创建数据库文件。随着 ADO 的不断发展,这些功能将会逐渐得到完善。

Visual J++ 包括一组封装 ADO 数据访问对象及允许 WFC 应用程序访问数

据的控件的 Java 类,这组类就是我们所说的“Java ActiveX 数据类”(ActiveX Data Objects for Java,ADOJ),它们位于 WFC 框架的 `com.ms.wfc.data` 和 `com.ms.wfc.data.ui` 数据包中。使用 ADOJ 的类及控件,可以连接到 OLE DB 数据提供者,操纵记录集,以及将记录集的字段绑定到窗体或 DHTML 页面 WFC 控件的属性中。为了协助访问 Windows 应用程序的数据,Visual J++ 还包含 Data Form Wizard(数据窗体向导),它使您可以创建连接到数据库的窗体。

正如您所看到的,Visual J++ 为访问 WFC 应用程序中的数据提供了大量支持。在本章中,我们将研究 ADOJ 的类及控件来演示与应用程序的数据进行交互是多么的容易。我们从研究如何使用 `DataSource` 和 `DataBinder` 控件来执行窗体上的数据绑定开始,然后研究数据控件。我们将通过应用程序读取数据库的例子,来演示如何使用 Visual J++ 的 `DataGrid` 控件一次查看多个记录。之后介绍如何直接访问数据库。数据绑定对于简单的应用程序是非常好的,但是在很多情况下,需要使用数据绑定与数据访问代码的组合。在很多情况下,您也许会发现通过代码直接访问数据会更容易一些。最后我们用 Visual J++ 提供的数据库应用程序向导 `Data Form Wizard` 来创建数据库应用程序。

使用完本章的实例,您将理解如何使用 ADO 来访问应用程序中的数据。虽然我们已经涉及到一些 ADO 提供的功能,但是在您的应用程序中或许还需要其他一些功能。要了解这些功能的详细信息,建议阅读 Visual J++ 的参考文档。

11.2 数据控件与数据库绑定

本节我们讲解如何通过使用 `DataSource` 和 `DataBinder` 控件来实现窗体与数据库的连接,我们将构建一个家庭影碟/录像带管理应用程序。图 11.1 显示了该实例在运行时的外观。完成该实例后,您将会发现在应用程序中创建与数据库连接的窗体是非常容易的。

11.2.1 创建工程并添加窗体

使用 Empty Project 模板创建一个新工程,然后将其命名为“VideoManager”。将一个新窗体添加到工程中,并将其命名为“VideoManager.java”。将窗体的 `text` 属性设置为 Video Manager,将 `startPosition` 属性设置为 Center Screen,并将 `borderStyle` 属性设置为 Fixed Dialog。设置 `size` 属性,使 `x = 460`,`y = 340`。

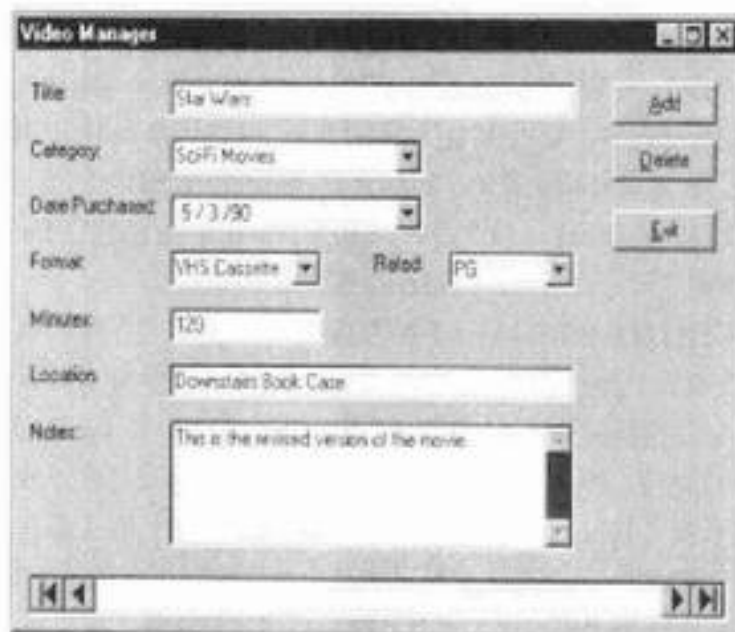


图 11.1 运行时的 Video Manager 实例

11.2.2 将控件添加到窗体中

我们需要将一些控件添加到窗体中。使用表 11.1 来添加控件并设置其属性。

表 11.1 VideoManager.java 组件及属性设置

组件类型	属性	设置
Label	Name	lblTitle
	Text	Title:
	TabIndex	0
Edit	Name	txtTitle
	Text	"
	TabIndex	1
Label	name	lblCategory
	text	Category:
	TabIndex	2

续表

组件类型	属性	设置
ComboBox	name	cmbCategory
	style	Dropdown
	text	"
	tabIndex	3
	items	Sci-Fi Movies Sports Related Movies Kids' Movies Action/Adventure Movies Historical Movies
Label	name	lblPurchase
	text	Date Purchased:
	tabIndex	4
DateTimePicker	name	dtpPurchase
	format	Short
	tabIndex	5
Label	name	lblFormat
	text	Format:
	tabIndex	6
ComboBox	name	cmbFormat
	style	Dropdown
	text	"
	tabIndex	7
	items	VHS Cassette Laser Disc DVD ROM
Label	name	lblRated
	text	Rated:
	tabIndex	8
ComboBox	name	cmbRated
	style	Dropdown
	text	"
	tabIndex	9
	items	G PG PG-13 R

续表

组件类型	属性	设置
Label	name	lblMinutes
	text	Minutes:
	tabIndex	10
Edit	name	txtMinutes
	text	...
	tabIndex	11
Label	name	lblLocation
	text	Location:
	tabIndex	12
Edit	name	txtLocation
	text	...
	tabIndex	13
Label	name	lblNotes
	text	Notes:
	tabIndex	14
Edit	name	txtNotes
	text	...
	multiline	true
	scrollBars	Vertical
	tabIndex	15
	tabIndex	16
Button	name	btnAdd
	text	&Add
	tabIndex	16
Button	name	btnDelete
	text	&Delete
	tabIndex	17
Button	name	btnExit
	text	&Exit
	tabIndex	18
DataSource	name	dsMain
	cursorType	Keyset
	designTimeData	true
	cursorLocation	Client
	lockType	Optimistic
	mode	ReadWrite

续表

组件类型	属性	设置
DataBinder	name	dbMain
	dataSource	dsMain
DataNavigator	name	DbMain
	dataSource	DbMain
	text	""
	tabIndex	21

图 11.2 显示了对话框在设计时的外观。

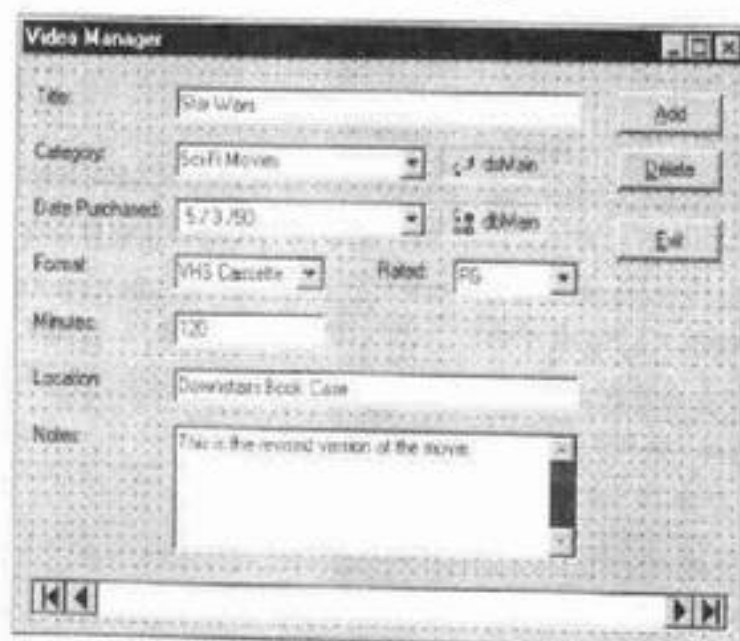


图 11.2 设计时的 VideoManager.java

除了 DataSource 控件之外,窗体的设计和属性设置是非常直观的。下面看一看设置的每一个属性。

- ◆ **cursorType** 属性允许您定义在 DataSource 控件的基本记录集中使用的光标类型。可以从 ForwardOnly、Keyset、Dynamic 和 Static 光标类型中进行选择。要了解如何设计使用光标的详细信息,请参见 Visual J++ 的参考文档。对于我们的实例,我们将采用 Keyset 光标。它允许我们更改记录集,并查看其他用户对数据库所做的潜在更改(不是添加)。
- ◆ **designTimeData** 属性可以确定记录集的记录是否显示在窗体的绑定控件中。在设计窗体时,这种功能对您会有所帮助,因为在设计窗体的布局时,它对于查看数据有时是很重要的。

- ◆ cursorLocation 属性定义光标的位置。通常情况下,对于本地数据源,应该将这个属性设置为 Client;对于远程数据源,应该将它设置为 Server。对于该实例,确保它被设置为 Client。
- ◆ lockType 属性允许您定义记录集记录的锁定方式。我们选择的是 Optimistic 记录锁定类型,因为我们没有共享信息。如果您正在使用共享数据源,那么您很可能会使用 Pessimistic 锁定方式。
- ◆ mode 属性允许您定义记录集记录的共享级别。这里,我们将使用 Readwrite 方式,以便我们可以读写记录集中的信息。

11.2.3 将 DataSource 控件与数据库联系起来

现在我们已经设置了窗体的大多数属性和控件,下面来设置 DataSource 控件,以便它可以连接到数据库中。我们在该实例中使用的数据库可以从本书附带的光盘中获得,它的名称为“Videos.mdb”。要将 DataSource 控件与数据库联系起来,可以单击窗体上的 dsMain 控件,然后单击 connectionString 属性。这时,在该属性的值区将显示一个 Ellipsis 按钮。单击该按钮,显示 Data Link Properties 页。

单击 Provider 选项卡,显示系统中的 OLE DB 提供者列表。图 11.3 显示了该选项卡的外观。

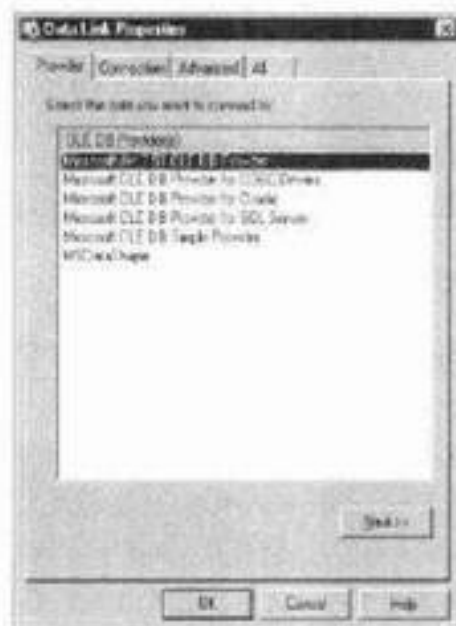


图 11.3 Data Link Properties 页的 Providers 选项卡

因为我们使用的是 Access 数据库,所以我们需要选择 Microsoft Jet 3.51 OLE

DB Provider, 然后单击 Connection 选项卡。在该选项卡中, 选择要连接到的数据库。如果我们选择的是其他数据库, 如 SQL Server, 那么 Connection 选项卡将提供选择登录到的服务器的选项、应用的安全性类型, 以及要连接到的数据库。浏览找到 Videos.mdb 文件, 单击 Test Connection 按钮, 确保有一个有效的链接。因为使用的是公共的 Microsoft Access 数据库, 所以不需要指定用户名及密码。该选项卡中其他选项不适合我们, 所以选定数据库后, 单击 OK 按钮。

如果想发布数据库应用程序, 那么您或许会考虑在包含 DataSource 控件的窗体的构造函数中, 手工编写 connectionString 属性的代码。Microsoft 存储本地数据库的习惯是将文件放在 \Windows\Application Data 下的目录中。您可以使用 SystemInformation 类的 getSpecialFolderPath 方法来获得当前用户 Application Data 目录的路径。



注意 在没有安装 Active Desktop 的 Windows 95/98 下使用时, getSpecialFolderPath 方法将返回一个空字符串。要使 getSpecialFolderPath 方法正常工作, 必须运行在包含 Active Desktop 的 Windows 98 或 Windows 95 下。

要发布数据库应用程序, 所要做的就是确保(或应用程序的安装程序)在用户 Application Data 目录中创建适当的目录, 然后将数据库复制到该路径中。以下是为 VideoManager 实例设置连接字符串的代码实例:



```
dsMain.setConnectionString(
    "Provider=Microsoft.Jet.OLEDB.3.51;"
    + "Persist Security Info=False;"
    + "Data Source="
    + SystemInformation.getSpecialFolderPath(
        SpecialFolder.APPLICATION_DATA)
    + "\\DevWorkshop\\VideoManager\\Videos.mdb");
```

在本书附带光盘实例的注释中, 以及在本书使用数据库的其他实例源代码的类似代码中, 包括了这个源代码。要在这些实例中使用该代码, 只需取消它的注释, 在 \Windows\Application Data 目录下创建指定的应用程序数据目录, 然后将实例的数据库文件复制到该路径下。因为设置连接字符串的代码位于 initForm 调用后的构造函数中, 所以代码将覆盖 initForm 方法中的 setConnectionString 调用。但是要记住, 在发布数据库应用程序时, 删除 initForm 方法中的所有 setConnectionString 调用, 只允许构造函数的 setConnectionString 调用来处理连接。

这时, 我们还没有任何记录, 因为还没有告诉 DataSource 控件我们要使用的表或准则。要指定 DataSource 控件的记录集, 可以单击 commandText 属性, 它允许您指定发送到 DataSource 控件的命令。通常情况下, 这是某些类型的 SQL 命令。可以通过更改 commandType 属性来选择要发送的命令类型。在我们的实例

中,我们将输入一个选择要显示的字段的较长 SQL 查询(我们希望显示除用作表唯一索引的 VideoID 属性之外的所有字段)。将下面的文本输入到 commandText 属性中:

```
select Category, Title, DatePurchased, Format, Rated, Minutes, Location, Notes FROM Videos ORDER by VideoID
```

一旦在属性中按 Enter 键或取消选中 commandText 属性,DataSource 控件包含的记录集就会填写指定的记录。如果控件是数据绑定的,那么它们将显示表中的第一个记录。

11.2.4 将控件绑定到数据库中

现在我们已经为 DataSource 控件创建了有效的记录集,下面把数据绑定到窗体的控件上。要做到这一点,我们将使用 DataBinder 控件。右击窗体上的 dbMain 控件,然后从显示的环境菜单中选择 Properties。这时将显示该控件的 Properties 页。图 11.4 显示了 Properties 对话框的外观。

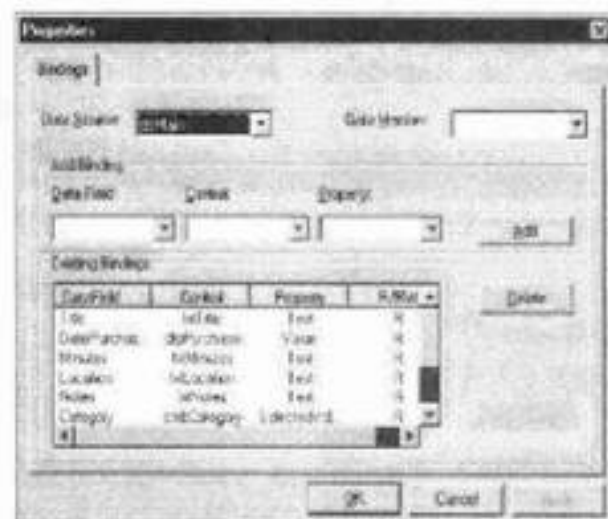


图 11.4 DataBinder 控件的 Properties 页

Properties 页非常容易使用。确保在 Data Source 组合框中选定有效的 DataSource 控件。要添加新的绑定,从 Data Field 下拉列表中选择要绑定的字段,从 Control 组合框中选择要将该字段绑定到的控件,然后从 Property 组合框中选择要绑定到字段中的控件的属性。根据选定的控件,要绑定的属性可能是不可用的。要选择列表中没有的属性,可以输入在 Forms Designer 的 Properties 窗口中显示的属性名称。设置好绑定后,单击 Add 按钮,将该绑定添加到 Existing Bindings 表。

继续这个过程,直到绑定完所有的控件。使用表 11.2 设置实例中的控件绑定。

表 11.2 VideoManager.java 的数据绑定

字段	控件	属性
Title	txtTitle	text
DatePurchased	dtpPurchase	value
Minutes	txtMinutes	text
Location	txtLocation	text
Notes	txtNotes	text
Category	cmbCategory	selectedIndex
Format	cmbFormat	selectedIndex
Rated	cmbRated	selectedIndex

您需要手工输入 ComboBox 控件的属性,因为它们在可绑定的属性列表中是不可用的。ADOJ 没有为 DataBinder 控件的 Properties 页提供 DataMember 组合框的完全支持。在将来的版本中,这个功能将得到完全的支持。dataMember 属性允许定义要使用的记录集。其他支持 ADO 的语言能够在一个 DataSource 控件中存储多个记录集。输入完所有的绑定后,单击 OK 按钮。当返回到 Forms Designer 时,注意我们的窗体显示的记录信息!甚至可以单击 DataNavigator 控件上的按钮来更改记录,因为我们将控件连接到了 dsMain 中。

11.2.5 为按钮添加事件处理程序及代码

完成数据绑定后,实例基本上就完成了。下面我们通过使用每个按钮的 click 事件默认的处理程序,来为我们添加到窗体中的 3 个按钮添加 click 事件处理程序。添加完事件处理程序后,将下面的代码添加到每一个事件处理程序中:



```
private void btnAdd_Click(Object source, EventArgs e)
{
    //判断数据库的当前的存取模式——ADD or UPDATE
    if (dsMain.GetRecordset().GetEditMode()
        == ADOEnums.EditMode.ADD)
    {
        try
        {
            //提交绑定控件的变化给记录集
            dsMain.CommitChanges();
            //更新数据库
            dsMain.GetRecordset().Update();
            //移动到最后一条记录
        }
        catch { }
    }
}
```

```
        dsMain.getRecordset().moveLast();
        //将该按钮重置为 Add 按钮
        btnAdd.setText("&Add");
        //重置 Delete 按钮
        btnDelete.setText("&Delete");
    }
    catch (Exception ex)
    {
        handleADOException(ex);
        dsMain.getRecordset().cancelUpdate();
    }
}
else
{
    try
    {
        //因为不在 ADD 模式下,所以必须取消当前记录上的任何改动
        dsMain.getRecordset().cancelUpdate();
        //进入 ADD 模式
        dsMain.getRecordset().addNew();
        //将该按钮重置为 Update 按钮
        btnAdd.setText("&Update");
        //将 Delete 按钮改为 Cancel 按钮
        btnDelete.setText("&Cancel");
    }
    catch (Exception ex)
    {
        handleADOException(ex);
    }
}

private void btnDelete_Click(Object source, Event e)
{
    try
    {
        //判断是否在 ADD 模式下,不是则删除当前记录
        if( dsMain.getRecordset().getEditMode()
            == AdoEnums.EditMode.ADD )
        {
            //取消 ADDNEW 过程
            dsMain.getRecordset().cancelUpdate();
            //将按钮重置为 Delete 按钮
            btnDelete.setText("&Delete");
        }
        //将按钮重置为 Add 按钮
    }
}
```

```

        btnAdd.setText("&Add");
    }
    else
    {
        //删除当前记录
        dsMain.getRecordset().delete(
                                                    AdoEnums.Affect.CURRENT);
        //判断该记录是否为最后一条记录
        if( dsMain.getRecordset().getRecordCount() > 0 )
        {
            //移到删除记录的下一条记录
            dsMain.getRecordset().moveNext();
            //如果下一条记录有 EOF 标志,则移动到最后一条记录
            if( dsMain.getRecordset().getEOF() )
                dsMain.getRecordset().moveLast();
        }
    }
}
catch (Exception ex)
{
    //处理异常
    handleADOException(ex);
    //确认 ADD 模式被全部清除
    dsMain.getRecordset().cancelBatch();
}

private void btnExit_Click(Object source, Event e)
{
    Application.Exit();
}

void handleADOException(Exception e)
{
    e.printStackTrace();
    MessageBox.Show( e.ToString(), "Video Manager" );
}

```

虽然我们只将 3 个按钮添加到窗体中,但是我们提供了 5 个按钮的代码支持。btnAdd 按钮支持添加新记录的过程,并且它还支持将新记录更新到数据库中。按钮更改它的标题来反映它当前所处的模式。完成后,我们就不再需要只有在添加记录时才有用的专门的 Update 按钮。btnAdd_Click 事件处理程序的代码从检查记录集的 EditMode 开始。它使用 AdoEnums 类的枚举来验证记录集当前是否被设置为添加新记录。如果当前 EditMode 被设置为添加新记录,那么将

继续更新代码。

更新代码从通过调用 `DataBinder` 控件的 `commitChange` 方法将用户对绑定控件所做的更改提交到记录集中开始。要将在记录集中所做的更改 `commitChange` 加到实际的数据库文件中,代码将调用 `DataSource` 控件的记录集中的 `update` 方法。要访问 `DataSource` 控件的记录集,代码将频繁地调用 `DataSource` 控件的 `getRecordset` 方法。用户添加记录后,`DataSource` 控件将它作为最后一个记录放在记录集中。要看到这个记录,代码将调用 `moveLast` 方法。最后,代码将 `btnAdd` 和 `btnDelete` 按钮的 `text` 属性重新设置为它们正常的文本标题。

如果用户在 `EditMode` 没有处在 `ADD` 模式时单击了 `btnAdd` 按钮,那么代码将把模式设置为 `ADD`。`ADD` 模式的代码从通过调用 `cancelUpdate` 方法来取消所有前面没有确定的更新开始。如果用户更改了当前记录的值,但还没有提交这个更改,那么更新可以被取消。代码将调用 `DataSource` 控件的记录集中的 `addNew` 方法。代码最后通过更改 `btnAdd` 的文本来反映该按钮现在将要作为 `Update` 按钮使用。代码还将更改 `btnDelete` 按钮,因为在添加记录时它将作为 `Cancel` 按钮使用。当 `btnDelete` 按钮的标题更改为“Cancel”时,它将允许用户退出添加新记录的过程。在 `btnAdd` 按钮的这两个模式下,`btnAdd_click` 事件处理程序代码将调用 `try/catch` 块中数据库代码来处理所有可能被抛弃的 ADO 异常情况。

像 `btnAdd_click` 代码那样,`btnDelete_click` 事件处理程序的代码确定应用程序是否正在添加新记录。如果是,那么代码将调用 `cancelUpdate` 来取消添加的新记录,并将 `btnAdd` 和 `btnDelete` 按钮的文本更改回它们最初的状态。如果代码没在添加新记录,那么代码将调用 `Recordset` 类的 `delete` 方法来删除当前显示的记录。要确保应用程序显示有效的记录,代码将确定当前记录集中是否存在记录。因为我们已经删除了记录,所以记录集可能是空的。

如果记录集是空的,那么代码将调用 `moveNext` 方法来移动到我们刚删除的记录后面的一个记录。移动到下一个记录可能会迫使光标离开记录集的最后一个记录。要确定这一点,代码将查看当前记录是否有 `EOF`(End Of File)标记。如果当前记录有 `EOF` 标记,那么我们需要将光标移回最后一个记录,以便代码调用 `moveLast`。`moveLast` 方法强迫光标移动到记录集的最后一个记录。当在记录集中移动时,应该检查 `EOF` 和 `BOF`(Beginning Of File)标记,以确保查看的是有效记录。

该实例的代码非常简单,这要在一定程度上感谢数据绑定。数据绑定取代了我们在读取窗体控件记录时必须编写的大量代码。要查看该实例的实际效果,编译并运行工程。当它启动时,将看到与 `Forms Designer` 中显示的相同的数据。可以使用 `DataNavigator` 控件的按钮移动到第一个和最后一个记录(使用向左的双箭头按钮移动到第一个记录,使用向右的双箭头移动到最后一个记录)。

反复移动记录,注意控件中的数据随着数据库中电影条目变化的情况。更改其中一个控件的值,移动到其他记录中,然后使用 `DataNavigator` 控件移回到该记录。注意这种更改会被保持下来。另外,试着添加一个新记录,并注意按钮是如何更改为 `Update` 和 `Cancel` 按钮的。以这种方式重新使用这些按钮为用户提供更加美观的界面。正如您所看到的,该实例非常有价值,并且它不需要我们做太多的工作。虽然数据绑定是一个强大的功能,但它也有不足之处。在 11.4 节的实例中,我们将研究在不使用数据绑定的情况下如何使用 `DataSource` 控件。

`VideoManager` 是一个完整的应用程序。为了使它的适用范围更广,可以将下面的一些建议应用到实例中,来帮助您管理大量的家庭影碟/录像带。

- ◆ 为用户提供一种配置应用程序的表,使应用程序更加灵活。由于我们不关心选定的工程是什么(因为我们将索引存储到工程中),所以添加新工程会相对容易一些。只是不要对数据排序,因为它将使以前存储的索引失效。要避免这种索引问题,可以使用 `selectedItem` 属性来存储选定工程实际的文本。
- ◆ 如果家庭影碟/录像带数据库规模巨大,那么您或许希望将 `DataSource` 使用的记录缩小为一个更有限的记录集合。可以添加一个功能来通过电影级别、分类或格式来搜索记录。一旦确定要显示的记录类型后,只需更改 `DataSource` 控件的 `commandText` 属性,并调用其记录集的 `requery` 方法。

11.3 使用数据控件

除了提供使用 `DataSource` 和 `DataBinder` 控件访问 ADO 的方便强大的方法外,Visual J++ 还提供了允许同时显示多个记录的控件。这就是 `DataGrid` 控件,我们已经在前面的实例中看到过它。它不仅可以查看多个记录,还可以执行一些操作,例如直接在控件中添加、删除和更新记录等。

与其他只提供绑定属性的 WFC 控件不同,`DataGrid` 控件支持复杂的绑定,它可以与绑定字段进行更多的交互。`DataGrid` 控件提供了 `dataSource` 属性,它可以指定 `DataGrid` 将要绑定的与其交互的 `DataSource` 控件。`DataGrid` 控件还可以根据它绑定的 `DataSource` 控件中的可用字段来更改显示的列。这种功能非常强大,因为它避免了使用指定的特殊数据格式硬编码列标题的需要。如果想更多地控制网格显示信息的方式,那么还可以操纵列。

在本节中,我们将通过把一个现成的 `PlayerExplorer` 实例引入到数据库应

用程序中,来研究 DataGrid 控件的功能。我们将使用 DataGrid 控件代替原来实例中的 ListView 控件显示运动员信息。图 11.5 显示了 Player Explorer 新版本在运行时的外观。

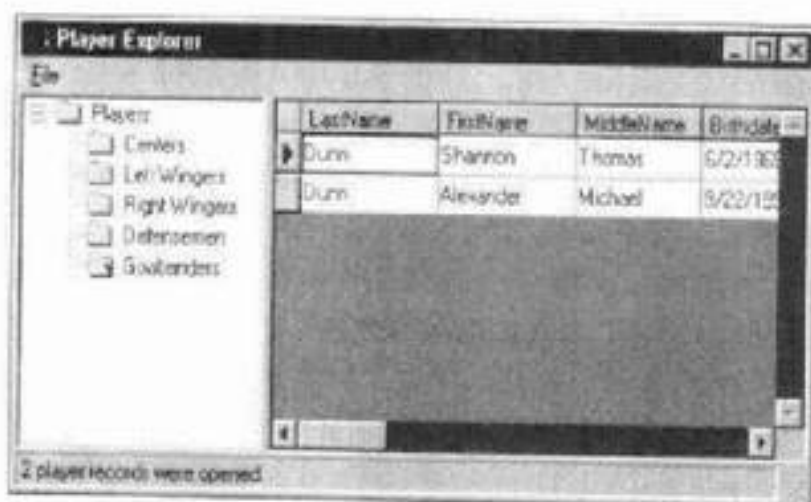


图 11.5 运行时的 PlayerExplorerDB 应用程序

还可以添加直接在 DataGrid 控件中编辑和删除运动员记录的能力。虽然可以在 DataGrid 控件中添加新记录,但是我们将禁用这个支持,以使更有用的 HockeyScoutDB 实例能够执行创建运动员记录的任务。完成该实例后,您将理解如何使用 DataGrid 控件来适应应用程序的需要。

11.3.1 新建工程

首先,可以从本书附带的光盘中获得现成的 PlayerExplorer 工程 (Samples\Chap11\PlayerExplorer 目录下),这个工程是利用 ListView 和 OpenFileDialog 控件来显示运动员的信息,根据 PlayerExplorer 工程创建一个新的工程。如果没有创建这个实例,那么将原工程的文件复制到名为“PlayerExplorerDB”的新目录中,然后在 Visual J++ 中打开该工程。

在添加新代码之前,我们需要通过删除一些原来用于解析运动员文件数据的代码,并删除 ListView 控件,来将这个应用程序转换成实际的数据库应用程序。在 Forms Designer 中打开 PlayerExplorer.java 文件,然后删除 ListView 和 OpenFileDialog 控件。还要删除 mnuSelectDir 和 sepFile1 菜单项,因为我们不再要求用户选择他们想要查找运动员记录的目录。接下来,在代码编辑器中显示同一个文件。删除类中所有的私有成员变量,以及 GetPrivateProfileString 和 GetPrivateProfileInt 的声明。还要确保删除 @dll.import 注释。

删除 `SelectPlayerDirectory` 和 `CreatePlayerLists` 方法,因为我们不再需要使用 `List` 对象来确定对于指定位置要显示的记录。最后,删除所有 `LoadPlayerInfo` 和 `LoadPlayers` 方法的代码,因为我们将替换它们的执行代码。

11.3.2 更改 `TreeView` 控件并添加 `DataSource` 和 `DataGrid` 控件

我们还需要为 `trvGroups` 控件的位置节点添加父节点。因为不必操纵列表和解析文件目录,所以这是一个非常简单的添加过程。使用 `nodes` 属性编辑器,添加一个名为“Players”的根节点。在 `Players` 节点中,为每一个位置添加一个子节点,删除原来的节点。

接下来,添加用来将 `DataGrid` 控件与 `HockeyScout.mdb` 文件联系起来的 `DataSource` 控件。将这个新的 `DataSource` 控件命名为“`dsMain`”。设置该控件,连接到 `Microsoft Jet 3.51 OLE DB Provider`,并将它的数据库设置为可以从本书附带光盘获得的 `HockeyScout.mdb` 文件。保留控件的 `commandText` 属性为空,因为我们将通过代码来更改它。设置 `cursorLocation`、`cursorType` 和 `mode` 属性,创建一个位于 `ReadWrite` 访问的 `Client` 中的键集光标。最后,设置 `DataSource` 的 `lockType` 属性,以使用 `Optimistic` 锁定类型。

因为删除了 `ListView` 控件,所以我们需要使用 `DataGrid` 控件来替代它。将 `DataGrid` 控件添加到窗体中,将它放在 `Splitter` 控件的右边,并将它的 `dock` 属性设置为 `Fill`,以使 `DataGrid` 控件占据屏幕其余的空间。将它命名为“`dbgPInfo`”。将 `DataGrid` 控件的 `allowAddNew` 属性设置为 `false`,保留其他的 `allowXXX` 属性为默认值。要将 `DataGrid` 控件绑定到 `DataSource` 控件,将它的 `dataSource` 属性设置为 `dsMain`。

原来的 `PlayerExplorer` 实例可以在运行时创建它的位置指定的列,以便只显示与位置相关的信息。在 `DataGrid` 控件中做到这一点要容易得多。要根据 `DataSource` 控件中的可用字段更改 `DataGrid` 控件的列,将 `dynamicColumns` 属性设置为 `true`。这时,在 `DataGrid` 控件中将看不到任何记录,因为我们没有定义 `DataSource` 控件的 `commandText`。另外,在用户选定它们要查看其记录的位置时,我们将通过代码来执行这种操作。

11.3.3 将代码添加到实例中

现在我们已经完成了新 `PlayerExplorer` 的用户界面部分,下面添加查询数据库和创建通过 `DataGrid` 控件显示的记录集的支持代码。将下面的代码添加到 `PlayerExplorer.java` 文件中:



```
private final String CENTER = "SELECT Players.LastName, "
    + "Players.FirstName, "
    + "Players.MiddleName, "
    + "Players.Birthdate, "
    + "Players.Birthplace, "
    + "Players.Height, "
    + "Players.Weight, "
    + "Players.College, "
    + "Players.CollegeName, "
    + "Players.ShotType, "
    + "Players.ShotAccuracy, "
    + "Players.PassShootRatio, "
    + "Players.Position "
    + "FROM Players "
    + "WHERE (((Players.Position) = "
    + "'Center'))";

private final String LEFTWING = "SELECT Players.LastName, "
    + "Players.FirstName, "
    + "Players.MiddleName, "
    + "Players.Birthdate, "
    + "Players.Birthplace, "
    + "Players.Height, "
    + "Players.Weight, "
    + "Players.College, "
    + "Players.CollegeName, "
    + "Players.ShotType, "
    + "Players.ShotAccuracy, "
    + "Players.PassShootRatio, "
    + "Players.Position "
    + "FROM Players "
    + "WHERE (((Players.Position) = "
    + "'Left Wing'))";

private final String RIGHTWING = "SELECT Players.LastName, "
    + "Players.FirstName, "
    + "Players.MiddleName, "
    + "Players.Birthdate, "
    + "Players.Birthplace, "
    + "Players.Height, "
    + "Players.Weight, "
    + "Players.College, "
    + "Players.CollegeName, "
    + "Players.ShotType, "
    + "Players.ShotAccuracy, "
```

```

+ "Players.PassShootRatio,"
+ "Players.Position"
+ "FROM Players"
+ "WHERE (((Players.Position) = "
+ "Right Wing'))";

private final String DEFENSE = "SELECT Players.LastName, "
+ "Players.FirstName, "
+ "Players.MiddleName, "
+ "Players.Birthdate, "
+ "Players.Birthplace, "
+ "Players.Height, "
+ "Players.Weight, "
+ "Players.College, "
+ "Players.CollegeName, "
+ "Players.ShotType, "
+ "Players.ShotAccuracy, "
+ "Players.PassShootRatio, "
+ "Players.Position"
+ "FROM Players"
+ "WHERE (((Players.Position) = "
+ "Defense'))";

private final String GOALIE = "SELECT Players.LastName, "
+ "Players.FirstName, "
+ "Players.MiddleName, "
+ "Players.Birthdate, "
+ "Players.Birthplace, "
+ "Players.Height, "
+ "Players.Weight, "
+ "Players.College, "
+ "Players.CollegeName, "
+ "Players.GloveSide, "
+ "Players.SkillLocations"
+ "FROM Players"
+ "WHERE (((Players.Position) = "
+ "Goaltender'))";

private void LoadPlayerInfo(Object source, TreeViewEvent e)
{
    TreeNode selNode = trvGroups.getSelectedNode();
    LoadPlayers(selNode);
}

//在 TreeView 控件选择的基础上向 ListView 控件载入数据

```

```
private void LoadPlayers(TreeNode selNode)
{
    //判断哪个节点被选中,指定合适的查询
    if (selNode.getText().equals("Centers"))
    {
        dsMain.setCommandText(CENTER);
        //报告用户有多少记录被打开
        stbMain.getStatusBarPanel(0).setText(
            dsMain.getRecordset().getRecordCount()
            + " player records were opened.");
    }
    else if (selNode.getText().equals("Left Wingers"))
    {
        dsMain.setCommandText(LEFTWING);
        //报告用户有多少记录被打开
        stbMain.getStatusBarPanel(0).setText(
            dsMain.getRecordset().getRecordCount()
            + " player records were opened.");
    }
    else if (selNode.getText().equals("Right Wingers"))
    {
        dsMain.setCommandText(RIGHTWING);
        //报告用户有多少记录被打开
        stbMain.getStatusBarPanel(0).setText(
            dsMain.getRecordset().getRecordCount()
            + " player records were opened.");
    }
    else if (selNode.getText().equals("Defensemen"))
    {
        dsMain.setCommandText(DEFENSE);
        //报告用户有多少记录被打开
        stbMain.getStatusBarPanel(0).setText(
            dsMain.getRecordset().getRecordCount()
            + " player records were opened.");
    }
    else if (selNode.getText().equals("Goaltenders"))
    {
        dsMain.setCommandText(GOALIE);
        //报告用户有多少记录被打开
        stbMain.getStatusBarPanel(0).setText(
            dsMain.getRecordset().getRecordCount()
            + " player records were opened.");
    }
    else
        dsMain.setCommandText("");
}
```

我们添加的第一段代码可能是最重要的。私有成员变量定义分配到 DataSource 控件 commandText 属性的查询。每一个技术位置都有一个相关的查询。虽然 SQL 查询较大,但是它们只是简单的记录选择器。它们之所以这么大是因为它们必须筛选掉对于给定的位置不需要显示的字段。下面我们来看一看代码如何使用这些 private final 成员变量。

当用户在 TreeView 控件中选定节点(如 Centers 节点)时,将触发 LoadPlayerInfo 事件处理程序。该事件处理程序中的代码将创建一个引用 TreeView 控件中当前选定节点的 TreeNode 对象。该节点被传递给 LoadPlayers 方法。LoadPlayers 方法的代码确定选定的节点。代码确定运动员的位置后,它将使用包含选定位置适当查询的私有成员变量设置 dsMain 的 commandText 属性。自动设置 commandText 属性将会使 DataGrid 控件更改它的显示,以反映记录集中的变化。

如果用户没有在 TreeView 控件中选定有效的位置节点,那么 commandText 将被设置为“”来表示记录集是空的。代码指定 commandText 后,它将更新窗体的 StatusBar 控件来报告打开的记录数。

添加完代码,编译并运行实例。应用程序显示后,在 TreeView 控件中选择一个位置节点,注意 DataGrid 控件将加载该位置的记录。要从 DataGrid 控件中删除行,您可以在控件中选定行,然后按 Delete 键。DataGrid 控件将删除该行,然后更新记录集以反映所做的删除操作。试着通过选定要编辑的行和列,然后键入新的值,来更改运动员记录中的值。如果离开该行并转到新的记录,那么应用程序将自动进行更改。

11.4 直接访问数据库

正如前面实例所演示的,数据绑定对于访问数据及在控件中显示数据是非常有用的功能。这种几乎可以绑定到控件任何属性的能力使数据绑定成为真正灵活的工具,虽然在我们需要更好地访问数据库中的数据时它还不能完全满足。例如,如果正在使用的控件支持多个值(如 ListBox 控件),那么该如何将一个字段绑定到控件中呢?数据绑定还不能确定很多存储选择,例如检索需要解析的字符串等。正是这种数据获取及存储过程控件的需求,要求您直接使用 ADO 类的对象和方法。随着应用程序要求的不断增长,以及数据越来越复杂,理解 ADO 将使开发人员的工作变得更加轻松。

在本节中,我们将把第 9 章“Java 应用程序编程进阶”中的冰球运动员选拔应用程序引入到使用数据库应用程序中。图 11.6 显示了该应用程序在运行时

的新版本。

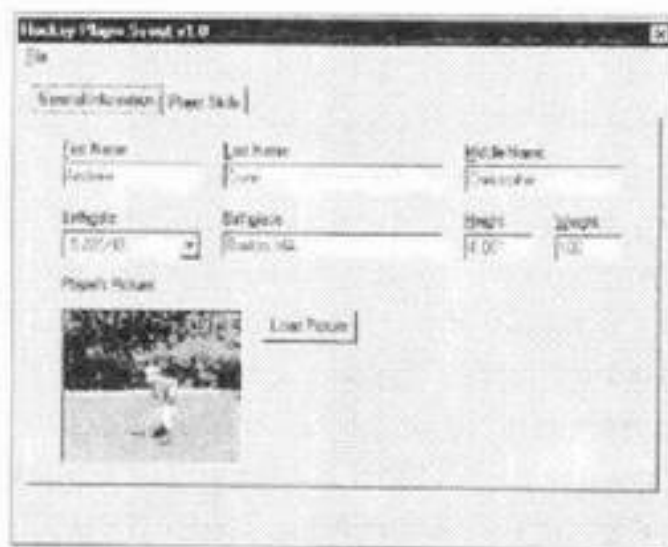


图 11.6 运行时的 HockeyScoutDB

因为在最初的实例中有各种控件,所以该应用程序非常适合于演示如何在没有数据绑定的情况下与数据库进行交互。在冰球运动员选拔应用程序的更新版本中,我们还可以重新使用一小部分原来的代码。我们添加的代码也许最初会令人畏惧,但是当完成该实例后,您将会方便地通过代码与数据库进行交互。

11.4.1 打开 HockeyScout 工程并准备代码

首先在第 9 章开发的 HockeyScout 应用程序基础上创建一个新工程。如果没有创建该实例,可以从本书附带的光盘中获得它(Samples\Chap09\HockeyPlayerScout 目录下)。将原来工程的文件复制到名为“HockeyScoutDB”的新目录中,然后在 Visual J++ 中打开该工程。

在添加新代码之前,我们必须将该程序转换成实际的数据库应用程序。需要删除一些原来创建指定选拔运动员记录文件的代码。在代码编辑器中打开 HockeyScout.java 文件,然后删除 SavePlayer 方法。新版本的 HockeyScout 应用程序只是将原来的应用程序引入到以数据库为中心的应用程序中。该增强版本允许在不关闭应用程序的情况下创建多个运动员记录。回顾原来的应用程序,用户在保存一个运动员记录后,将关闭应用程序。

另外,现在应用程序可以更新记录。该功能非常重要,因为原来的应用程序没有更新运动员数据的方法。在新版本中,我们将使用创建新运动员记录及更新现有记录的事件处理程序和帮助程序方法来代替我们删除的 SavePlayer 方法。该实例新版本中另一个重要的变化是它将使用菜单来代替使用底部的按钮。删

除 btnSave 和 btnClose Button 控件,将 MainMenu 控件添加到窗体中,并将该控件命名为“mnuMain”。调整窗体的大小,使其适合 TabControl 的大小,因为在窗体底部不再有 Button 控件。使用表 11.3 设计实例的菜单。

表 11.3 HockeyScout.java 菜单设计

标题	名称	父菜单
&File	MnuFile	none
&NewPlayerRecord	MnuNewPlayer	mnuFile
&UpdatePlayerRecord...	MnuUpdatePlayer	mnuFile
&SavePlayerRecord	MnuSavePlayer	mnuFile
< Separator >	SepFile1	mnuFile
E&xit	MnuExit	mnuFile

因为只有在创建运动员记录或更新现有运动员记录时才使用 mnuSavePlayer,所以将菜单项的 enabled 属性设置为 false。除了添加的菜单项,我们还需要为除 sepFile1 和 mnuFile 以外的所有 click 事件创建处理程序。对于 mnuNewPlayer 菜单项,创建一个 click 事件处理程序,并将其命名为“CreateNewPlayer”。对于 mnuSavePlayer 菜单项,创建一个 click 事件处理程序,并将其命名为“SaveData”。最后,将 mnuExit 菜单项与先前存在的 ExitApp 事件处理程序联系起来。

11.4.2 添加 DataSource 控件

定义完用户界面,下一步添加用来管理应用程序记录集的 DataSource 控件。将这个新的 DataSource 控件命名为“dsMain”。设置该控件,连接到 Microsoft Jet 3.51 OLE DB Provider,然后将它的数据库设置为附带光盘提供的 HockeyScout.mdb 文件。将 DataSource 控件的 commandText 属性设置为 Select * FROM Players。设置 cursorLocation、cursorType 和 mode 属性,来创建一个位于 Readwrite 访问的 Client 中的键集光标。另外,设置 DataSource 控件的 lockType 属性,以使用 Optimistic 锁定。

11.4.3 创建 PlayerSelect 对话框

我们将向用户提供更新存储在数据库中现有运动员记录的能力。为了更有效地做到这一点,我们提供一种选择要编辑的运动员记录的方法。将使用 Data Form Wizard 来创建一个显示数据库中所有可用运动员列表的对话框(参看 11.5 节)。从 Add Item 对话框中启动 Data Form Wizard。将新对话框命名为“PlayerSelect”。选定 Access 作为数据库类型,选定 HockeyScout.mdb 文件作为数据库名

称,然后将窗体布局设置为 Grid(Datasheet)。

在 Record Source 步骤中,选定 Players 表,然后将 LastName、FirstName 和 Position 字段添加到记录源中。在 Control Selection 对话框中,取消选定包括 DataNavigator 控件在内的所有按钮。我们将添加自己的按钮,因为我们不使用传统风格的窗体。完成向导,然后在 Forms Designer 中加载 PlayerSelect 窗体。将两个按钮添加到窗体中,然后使用图 11.7 设计窗体的布局。

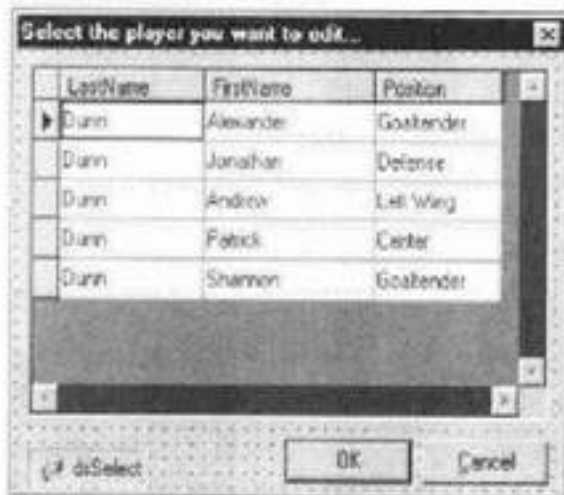


图 11.7 设计时的 PlayerSelect 对话框

将一个按钮命名为“btnOK”,并将其 text 属性设置为 OK。将另一个按钮命名为“btnCancel”,并将其 text 属性设置为“&Cancel”。对于对话框,我们将使用 dialogResult 属性。这个属性允许您定义按钮的默认行为,并使调用该对话框的线程知道在调用 Form 类的 getDialogResult 方法时用户单击的按钮。例如,把按钮的 dialogResult 属性设置为 OK,将会使对话框自动关闭,并且窗体的对话框结果将被设置为 DialogResult.OK。这样可以避免为诸如 Cancel 等按钮创建事件处理程序,其中您只需调用 dispose 方法来退出窗体。将 btnOK 控件的 dialogResult 属性设置为 OK;将 btnCancel 控件的 dialogResult 属性设置为 Cancel。

虽然我们已经为这两个按钮定义了对话框,但是我们仍需要为 btnOK 控件创建一个事件处理程序。为 btnOK 的 click 事件创建一个处理程序,并将它命名为“PlayerSelected”。将下面的代码添加到事件处理程序中,然后将成员变量和访问体方法添加到类中。



```
//在选中的记录上存储书签
```

```
private Object m_oSelPlayer = new Object();
```

```
//设置公共变量,用于存储选中的运动员
```

```
private void PlayerSelected(Object source, Event e)
```

```

//获取运动员的列表的选中行
Object[] tempSel = dataGrid.getSelectedRows();
//检查是否选中
if (tempSel.length > 0)
    //将选中的第一个运动员的书基指定给公共变量
    m_oSelPlayer = tempSel[0];
else
{
    //用户也许只选一个单元格而不是整个一行,选择当前行
    m_oSelPlayer = dataGrid.getCurrentRow();
}

```

PlayerSelected 事件处理程序的代码从创建 Object 阵列开始,然后通过调用 getSelectedRows 来使用 DataGrid 控件中当前选定的行填写它。该方法是必需的,因为网格中可以有多个选定的行。如果阵列没有返回空值(表明没有选定的行),那么代码将把阵列中的第一个工程分配给私有成员变量 m_oSelPlayer。对话框外的代码可以通过添加的 getSelectedPlayer 访问体访问这个私有变量。如果没有选定的行,那么代码将检索与 DataGrid 控件中的行(光标所在的位置)对应的记录,然后将该记录分配给 m_oSelPlayer。

在完成该对话框之前,我们还需要做一些更改。将窗体的 text 属性更改为 Select the player you want to edit。将它的 acceptButton 设置为 btnOK,然后将它的 cancelButton 设置为 btnCancel。因为这是一个对话框,所以还应该将窗体的 borderStyle 属性设置为 Fixed Dialog。为了防止无意中更改了 DataGrid 控件中的记录,可以将控件中除 allowRowSizing 属性之外的所有其他 allowXXX 属性设置为 false。将 allowRowSizing 属性设置为 true。

11.4.4 添加访问数据库的代码

完成 PlayerSelect 窗体后,下面我们将把代码添加到 HockeyScout 类中,以便我们能够从数据库读入到控件中,反之亦然。将下面的代码添加到 HockeyScout.java 文件中。然后,我们将介绍代码重要的部分,以了解它工作的方式及程序通常的流程。



```

private int m_iDataMode = ADOEnums.EditMode.NONE;
//用于存储当前要被更新的运动员

private Object m_oCurPlayer = new Object();

```

//重置窗体上的所有控件

```
private void ResetControls()
```

```
{
```

```
    //重置 General Information 标签页
```

```
    ResetTabPage(tpGeneral);
```

```
    //重置 Skills Information 标签页
```

```
    ResetTabPage(tpSkills);
```

```
}
```

//重置指定标签页上的所有控件,以使新数据能被置入

```
private void ResetTabPage(TabPage tpTemp)
```

```
{
```

```
    for (int i = 0; i < tpTemp.getControlCount(); i++)
```

```
    {
```

```
        Control tempCtrl = tpTemp.getControl(i);
```

```
        if (tempCtrl.getClass() == Edit.class)
```

```
        {
```

```
            tempCtrl.setText("");
```

```
        }
```

```
        else if (tempCtrl.getClass() == DateTimePicker.class)
```

```
        {
```

```
            DateTimePicker tempPicker = (DateTimePicker)tempCtrl;
```

```
            tempPicker.setValue(new Time());
```

```
        }
```

```
        else if (tempCtrl.getClass() == PictureBox.class)
```

```
        {
```

```
            PictureBox tempPic = (PictureBox)tempCtrl;
```

```
            tempPic.setImage(null);
```

```
        }
```

```
        else if (tempCtrl.getClass() == ComboBox.class)
```

```
        {
```

```
            ComboBox tempCombo = (ComboBox)tempCtrl;
```

```
            tempCombo.setSelectedIndex(-1);
```

```
        }
```

```
        else if (tempCtrl.getClass() == ListBox.class)
```

```
        {
```

```
            ListBox tempList = (ListBox)tempCtrl;
```

```
            tempList.setSelectedIndex(-1);
```

```
        }
```

```
        else if (tempCtrl.getClass() == UpDown.class)
```

```
        {
```

```
            UpDown tempUD = (UpDown)tempCtrl;
```

```
            tempUD.setValue(0);
```

```
        }
```

```

else if (tempCtrl.getClass() == CheckedListBox.class)
{
    CheckedListBox tempList = (CheckedListBox) tempCtrl;
    for (int x = 0; x < tempList.getItemCount(); x++)
    {
        tempList.setItemCheck(x, false);
    }
}
else if (tempCtrl.getClass() == RadioButton.class)
{
    RadioButton tempOpt = (RadioButton) tempCtrl;
    if (tempOpt.getName() == "optLeft")
        tempOpt.setChecked(true);
}
else if (tempCtrl.getClass() == CheckBox.class)
{
    CheckBox tempCheck = (CheckBox) tempCtrl;
    tempCheck.setChecked(false);
}
else if (tempCtrl.getClass() == TrackBar.class)
{
    TrackBar tempTrack = (TrackBar) tempCtrl;
    tempTrack.setValue(5);
}
}
}

//处理 New Player 菜单项,新建一个运动员记录
private void CreateNewPlayer(Object source, Event e)
{
    ResetControls();
    m_iDataMode = AdoEnums.EditMode.ADD;
    mnuSavePlayer.setEnabled(true);
    mnuNewPlayer.setEnabled(false);
    mnuUpdatePlayer.setEnabled(false);
}

//处理 Update Player 菜单项,编辑一个已有的运动员记录
private void UpdatePlayer(Object source, Event e)
{
    PlayerSelect dlgSelect = new PlayerSelect();
    dlgSelect.setVisible(false);
    dlgSelect.ShowDialog();

    //判断对话框中的 OK 按钮是否被按下

```

```

        if (dlgSelect.getDialogResult() == DialogResult.OK)
        {
            m_oCurPlayer = dlgSelect.getSelectedPlayer();
            LoadControlData();
            mnuSavePlayer.setEnabled(true);
            mnuUpdatePlayer.setEnabled(false);
            mnuNewPlayer.setEnabled(false);
            m_iDataMode = AdoEnums.EditMode.INPROGRESS;
        }
    }

    //从当前记录集载入数据到各个控件
    private void LoadControlData()
    {
        if (dsMain.getRecordset() != null)
        {
            //获得当前记录集
            Recordset tempRs = dsMain.getRecordset();
            //移动到选中的运动员记录
            tempRs.move(0, m_oCurPlayer);
            //载入 First Name
            txtFirstName.setText(
                tempRs.getField("FirstName").getString());
            //载入 Last Name
            txtLastName.setText(
                tempRs.getField("LastName").getString());
            //载入 Middle Name
            txtMiddleName.setText(
                tempRs.getField("MiddleName").getString());
            //载入 Birthdate
            dtpBDay.SetValue(new Time(tempRs.getField("Birthdate")
                .getString()));
            //载入 Birthplace
            txtBirthplace.setText(
                tempRs.getField("Birthplace").getString());
            //载入 Height
            txtHeight.setText(
                tempRs.getField("Height").getString());
            //载入 Weight
            txtWeight.setText(
                tempRs.getField("Weight").getString());
            //载入运动员图片
            m_sPicturePath = tempRs.getField("PlayerPicture")
                .getString();
            if (m_sPicturePath != null)

```

```

    try
    {
        picPicture.setImage(new Bitmap(m_sPicturePath));
    }
    catch (Exception ex)
    {
        MessageBox.show("Invalid filename for picture.",
                        m_sAppName,
                        MessageBox.ICONERROR);
        return;
    }
}

//载入 Position
cmbPosition.setSelectedItem(tempRs.getField("Position")
                           .getString());
//载入 College Yes or No
chkCollege.setChecked(tempRs.getField("College")
                      .getBoolean());
//载入 College Name
txtCollege.setText(tempRs.getField("CollegeName")
                  .getString());

if ((String)cmbPosition.getSelectedItem()
    == "Goaltender")
{
    if (tempRs.getField("GloveSide").getString()
        .equals("Left"))
        optLeft.setChecked(true);
    else
        optRight.setChecked(true);

    String tempSkills = tempRs
                        .getField("SkillLocations")
                        .getString();

    while (tempSkills.length() != 0)
    {
        String tempString;
        int index = tempSkills.indexOf(",");
        if (index > 0)
            tempString = tempSkills.substring(0, index);
        else
            tempString = tempSkills.trim();
    }
}

```

```

        for (int i = 0;
            i < clstSkillLoc.getItemCount(); i++)
        {
            if (clstSkillLoc.getItem(i).toString()
                .equals(tempString))
            {
                clstSkillLoc.setItemCheck(i, true);
                break;
            }
        }

        tempSkills = tempSkills.substring(index + 2);
    }
    else
    {
        //载入 Shot Type
        lstShotType.setSelectedItem(
            tempRs.getField("ShotType").getString());
        //载入 PassShootRatio
        trbPassShoot.setValue(
            tempRs.getField("PassShootRatio").getInt());
        //载入 Shot Accuracy
        udShotAc.setValue(
            tempRs.getField("ShotAccuracy").getInt());
    }
    else
    {
        MessageBox.show("The recordset is empty!"
            + "Please verify the database's "
            + "location and that the player's "
            + "table is not empty",
            m_sAppName,
            MessageBox.ICONEXCLAMATION);
    }
}

//处理 Save Player 菜单项
private void SaveData(Object source, Event e)
{
    ManageData(m_iDataMode);
    mnwNewPlayer.setEnabled(true);
    mnwUpdatePlayer.setEnabled(true);
    mnwSavePlayer.setEnabled(false);
}

```



```
|  
  
//执行实际的操作来更新和新建运动员记录,同时在更新数据库之后,  
//调用 ResetControls 清除控件  
private void ManageData(int Mode)  
{  
    if (dsMain.getRecordset() != null)  
    {  
        //得到当前记录集  
        Recordset tempRs = dsMain.getRecordset();  
  
        //判断当前模式  
        if (Mode == ADOEnums.EditMode.ADD)  
        {  
            //进入 Add 模式  
            tempRs.addNew();  
        }  
        else  
        {  
            //移动到选中的记录  
            tempRs.move(0, m_oCurPlayer);  
        }  
  
        //更新 FirstName  
        tempRs.getField("FirstName")  
            .setString(txtFirstName.getText());  
        //更新 LastName  
        tempRs.getField("LastName")  
            .setString(txtLastName.getText());  
        //更新 MiddleName  
        tempRs.getField("MiddleName")  
            .setString(txtMiddleName.getText());  
        //更新 Birthdate  
        tempRs.getField("Birthdate")  
            .setString(dtpBDay.GetValue().ToString());  
        //更新 Birthplace  
        tempRs.getField("Birthplace")  
            .setString(txtBirthplace.getText());  
        //更新 Height  
        tempRs.getField("Height")  
            .setString(txtHeight.getText());  
        //更新 Weight  
        tempRs.getField("Weight")  
            .setString(txtWeight.getText());  
        //更新 Player Picture
```

```
tempRs.getField("PlayerPicture")
    .setString(m_sPicturePath);
//更新 Position
tempRs.getField("Position")
    .setString(cmbPosition.getText());
//更新 College
tempRs.getField("College")
    .setBoolean(chkCollege.getChecked());
//更新 CollegeName
if (chkCollege.getChecked())
    tempRs.getField("CollegeName")
        .setString(txtCollege.getText());

if (cmbPosition.getSelectedItem() == "Goaltender")
{
    //更新 Glove Side
    if (optLeft.getChecked() == true)
        tempRs.getField("GloveSide").setString("Left");
    else
        tempRs.getField("GloveSide").setString("Right");

    //更新 Skill Locations
    String combined = new String();
    //遍历 ListBox 中所有选中的项
    for (int i = 0; i < clstSkillLoc.getItemCount(); i++)
    {
        //把选中的项加入到字符串中
        if (clstSkillLoc.getItemChecked(i))
        {
            if (combined.length() == 0)
                combined = (String)
                    clstSkillLoc.getItem(i);
            else
                combined = combined + "," +
                    (String)
                        clstSkillLoc.getItem(i);
        }
    }
    //将合成的字符串加入到 SkillLocations 中
    tempRs.getField("SkillLocations")
        .setString(combined);
}
else
{
    //更新 Shot Type
```

```

tempRs.getField("ShotType")
    .setString((String)lstShotType
    .getSelectedItem());
//更新 Pass Shoot Ratio
tempRs.getField("PassShootRatio")
    .setInt(trbPassShoot.getValue());
//更新 Shot Accuracy
tempRs.getField("ShotAccuracy")
    .setInt(udShotAc.getValue());
|
//重置控件
ResetControls();
//更新数据库
tempRs.update();
//重置模式标志
m_iDataMode = AdoEnums.EditMode.NONE;

//为用户弹出显示操作结果的消息框
if (m_iDataMode == AdoEnums.EditMode.ADD)
    MessageBox.show("Player created successfully!",
        m_sAppName,
        MessageBox.ICONINFORMATION);
else
    MessageBox.show("Player updated successfully!",
        m_sAppName,
        MessageBox.ICONINFORMATION);
|
|

```

正如您所看到的,我们需要一些代码来提供数据绑定提供的大多数功能。主要的区别是我们有更多的事件控件,先来看一下我们声明的成员变量。`m_iDataMode` 属性存储我们当前所使用的数据模式,该模式将使代码能够确定它是在创建新记录还是在更新现有的记录,这使我们可以创建一个处理这两种情况的方法。声明的另一个成员变量 `m_oCurPlayer` 在更新现有运动员记录时使用,它将书签存储到当前编辑的运动员记录中。

下面我们先来看一下创建新运动员记录的过程,因为应用程序还需要更多的代码来更新运动员记录。当用户选择 `mnuNewPlayer` 菜单项时,将触发 `CreateNewPlayer` 事件处理程序,该事件处理程序的代码从调用我们添加的 `ResetControls` 方法开始。`ResetControls` 清除所有控件的内容,这是其中一个绑定控件自动为您处理的功能。`ResetControls` 的代码调用 `ResetTabPage` 方法来清除每个 `TabPage` 对象的控件,因为控件被包含在 `TabPage` 对象中,所以调用 `TabControl` 控件中每个 `TabPage` 对象的方法是很必要的。

`ResetTabPage` 方法使用 `for` 循环来遍历该方法参数指定的 `TabPage` 对象的所有控件。在 `for` 循环中,代码访问存储在指定索引的 `TabPage` 对象控件阵列中的控件。代码将确定一个当前控件的 `if/else if` 代码块。一旦有匹配的指定类,代码就使用当前控件作为基础来创建指定控件类的实例。有效控件引用后,代码可以调用类的指定方法来清除内容或重新设置指定控件的值。当您有很多控件,并且不希望创建窗体控件的固件控件引用时,这种处理窗体控件的方法是很有用的。代码还允许您将控件添加到 `if/else if` 块中指定类型的窗体中,并且代码会自动处理它们。

`ResetControls` 方法返回到 `CreateNewPlayer` 事件处理程序后,代码就使用 `AdoEnums.EditMode` 枚举类设置 `m_iDataMode` 标记,来指出我们正在创建新的运动员记录。代码启用 `mnuSavePlayer` 菜单项,以便用户可以保存它们的运动员记录。代码将禁用 `mnuUpdatePlayer` 和 `mnuNewPlayer` 菜单项以防止重复输入。然后用户继续在对话框的控件中输入值。

当完成运动员信息的输入后,用户将选择 `mnuSavePlayer` 菜单项。该操作将触发 `SaveData` 事件处理程序。`SaveData` 从调用 `ManageData` 方法开始。`ManageData` 根据用户是创建还是更新运动员记录来更新数据库。`ManageData` 的代码从查看 `dsMain` 中是否存在有效的 `Recordset` 对象开始。这种检查是预先警告的,并确保有一个有效的地方来保存运动员的信息。如果有效的 `Recordset` 对象是可用的,那么代码将创建 `dsMain` 记录集的引用。

具有记录集的有效引用后,代码将查看我们是否在创建运动员记录。如果是,那么它将调用 `Recordset` 的 `add` 方法来通知 `Recordset` 对象正在创建新记录。否则,代码将通过调用 `Recordset` 对象中的 `move` 方法来通知 `Recordset` 对象用户在 `PlayerSelect` 对话框中选定的记录。您或许会想起,`m_oCurPlayer` 对象存储我们正在更新的运动员记录的标签,在这部分代码之后,其余的代码对于新记录和更新的记录都是相同的。

代码通过使用 `tempRs` 的 `getField` 方法来检索数据库中的每一个字段,然后它将调用适当的 `set` 方法,使用其相关控件中存储的数据来设置字段的值。这些 `set` 方法使存储正确字段的数据类型很容易。如果我们不知道字段相应的数据类型,那么通常我们可以使用 `setValue` 方法,它接受可以存储任意类型数据的 `com.ms.com.Variant` 对象。

代码继续统计所有的字段,直到它到达运动员位置特有的字段。然后它查看当前运动员是否为守门员。如果是,那么代码将设置与守门员信息相关的字段;否则,与前锋相关的信息将被存储到适当的字段中。

查看这些代码后,您或许会问自己一些问题。为什么使用一个表来存储两种类型的所有信息呢?为什么没有使用相关的数据库呢?为什么存储字符串作

为 Glove Hand 选项按钮的值呢? 同样的答案适用于所有这些问题: PlayerExplorer 实例。在下一个实例中,我们将采用 PlayerExplorer 应用程序(用于显示运动员记录),并将它转换成数据库。这个新版本将不使用标准的 ListView 控件,但是它将使用 DataGrid 控件。为了使 DataGrid 控件中显示的数据更易于绑定且更有意义,最好用数据库存储数据,以便 PlayerExplorer 可以很容易地查看它。

代码更新指定的字段后,它将调用 ResetControls 方法来重新设置对话框,然后调用 Recordset 对象中的 update 方法。从前面的实例您可能会想起,update 方法将 Recordset 对象的更改提交到基本的数据库文件中。代码更新数据库后,MessageBox 对象将通知用户记录已经成功地被创建或更新。代码重新设置 m_i-DataMode 标记。一旦 ManageData 方法将控制返回到 SaveData 事件处理程序时,代码就以重新设置菜单而结束。

下面我们来看一看当用户更新现有运动员记录时进行的操作。用户选择 mnuUpdatePlayer 菜单项,将触发 UpdatePlayer 事件处理程序。UpdatePlayer 的代码从创建 PlayerSelect 对话框的实例并显示它开始。用户从对话框中选定要编辑的运动员记录并单击对话框中的 OK 按钮时,代码将通过调用 getDialogResult 方法来检查对话框的结果。如果用户单击 OK 按钮,那么代码将调用 PlayerSelect 对话框的 getSelectedPlayer 方法,它将返回用户选定的运动员的书签。该书签存储在 m_oCurPlayer 成员变量中。

接下来,UpdatePlayer 调用 LoadControlData 方法来将运动员的信息加载到控件中。大多数情况下 LoadControlData 与 ManageData 方法相反,它用来将信息从控件存储到数据库中。一旦 LoadControlData 从加载的控件返回数据,UpdatePlayer 中的代码将禁用 mnuUpdatePlayer 和 mnuAddPlayer 菜单项,并启用 mnuSavePlayer 菜单项。代码还将当前的数据模式设置为 AdoEnums.EditMode.INPROGRESS 值,以指出我们正在更新运动员记录,而不是在创建新的运动员记录。此后,代码和处理过程与创建新的运动员记录是相同的。

编译并运行实例,试着创建和更新运动员记录。该应用程序已经变得比我们最初创建时有用的多。如果需要,可以更改窗体标题栏中的应用程序版本号,以适应我们添加的最新功能。

虽然我们添加了数据库支持,提供在不关闭应用程序的情况下创建新运动员记录的能力,以及更新了运动员记录,但是我们仍可以添加其他功能。以下是一些改进建议,可以用来使应用程序功能更加全面。

- ◆ 添加取消 New Player Record 或 Update Player Record 过程的能力。
- ◆ 删除记录的能力可能也很有用。

- ◆ 为使用最初的实例创建的现有运动员记录提供一个导入工具。通过读取运动员记录,设置带有数据的控件,然后调用使用的代码来保存新的运动员记录,可以在保存运动员记录时重新使用现有的代码。不要忘了设置 `m_i-DataMode` 标记,以便代码知道操作正在做什么。

11.5 使用 Data Form Wizard

除了前面几节介绍的方法,还可以使用 Data Form Wizard 来连接数据提供者,以及将数据绑定到控件中。虽然该向导生成的代码不一定满足需要,但这是一个很好的 ADO 指南。一旦理解了如何在应用程序中使用 ADO,就会发现使用 Data Form Wizard 是一种开发连接到数据库的窗体的好方法。Data Form Wizard 可以创建连接到系统各种可用数据提供者的窗体。

在本章中,我们将创建一种更有意义的数据窗体(Master/Detail 窗体),同时研究各种 Data Form Wizard 选项。我们使用向导开发的实例将使用 Microsoft Access 附带的 Northwind 数据库。Northwind 也包含在本书附带的光盘中(Samples\Chap11\CustomerOrders 目录下)。向导生成的窗体将使用标准控件显示一个客户记录,并使用 DataGrid 控件显示客户订单。完成该实例后,您将理解 Data Form Wizard 怎样帮助您开发应用程序,以及如何充分地利用它。

11.5.1 创建工程并显示 Data Form Wizard

使用 Empty Project 模板创建一个空工程,然后将这个新工程命名为“CustomerOrders”。要打开 Data Form Wizard,可以从 Project 菜单选择 Add Form。这时将显示 Add Item 对话框。选定 Data Form Wizard,然后将窗体命名为“CustomerOrders”。单击 OK 按钮,启动向导。

Data Form Wizard 的第一步允许您加载包含向导在以前会话中存储的设置的文件。因为我们不想使用以前的设置,所以我们将配置文件设置为 None,然后单击 Next 按钮,进入下一个向导界面。

11.5.2 指定数据库类型及名称

在本书的数据库实例中,我们将使用 Microsoft Access,因为它可以比较方便地说明基本的数据库概念。因为 ADO 适应不同的数据库平台,所以可以很容易

地将这些实例导入到 Microsoft SQL Server 及其他可以通过开放式数据库连接 (ODBC) 访问的数据库平台中。

下一步, Data Form Wizard 要求您选择连接的数据库类型。有两种选择: Access 和 ODBC。图 11.8 显示了这一步向导的外观。

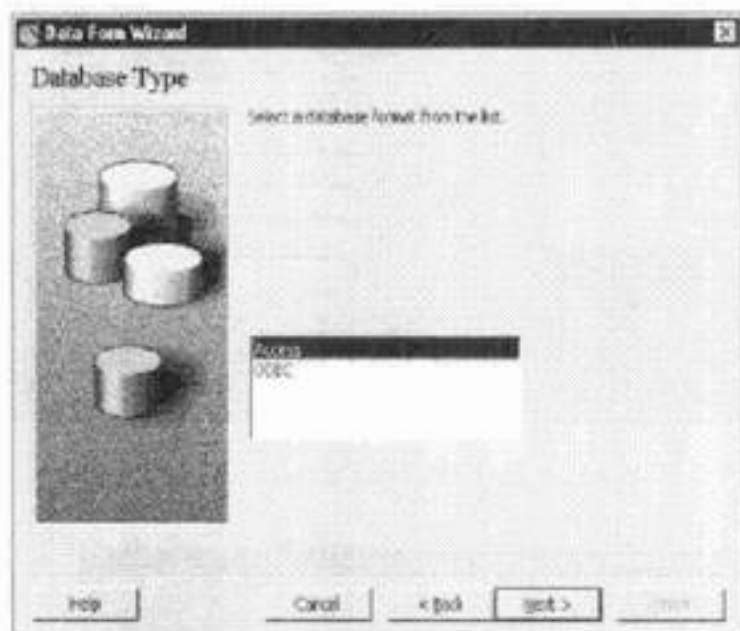


图 11.8 Data Form Wizard 的 Database Type 步骤

Access 选项指定“Microsoft Jet OLE DB 提供者”(直接连接到 Access 数据库的平台)作为数据平台; ODBC 选项允许您连接到 ODBC 数据库的数据提供者。ODBC 使您可以连接到系统中任意安装有 ODBC 驱动程序的数据库平台上。如果您选择的是 ODBC 选项, 那么下一步向导允许您选择要使用的数据源名称 (DSN)、用户名、密码及数据库名称。如果希望使用远程数据库, 那么可以选择驱动程序及要连接的、代替 DSN 的服务器。DSN 允许您选择在【控制面板】中的 ODBC 部分配置的数据库。图 11.9 显示了当选中 ODBC 选项时向导 Connect Information 步骤的外观。这里, 我们将使用 Access 数据库, 选中 Access 选项, 然后单击 Next 按钮。

因为我们选中的是 Access 选项, 所以接下来的向导将要求您指定要使用的数据库名称。图 11.10 显示了这一步的向导外观。可以通过单击 Browse... 按钮来浏览数据库。如果系统中有 Northwind 数据库, 那么可以选定它; 否则, 可以从随书光盘中选定它。选定 Northwind 数据库文件后, 单击 Next 按钮。

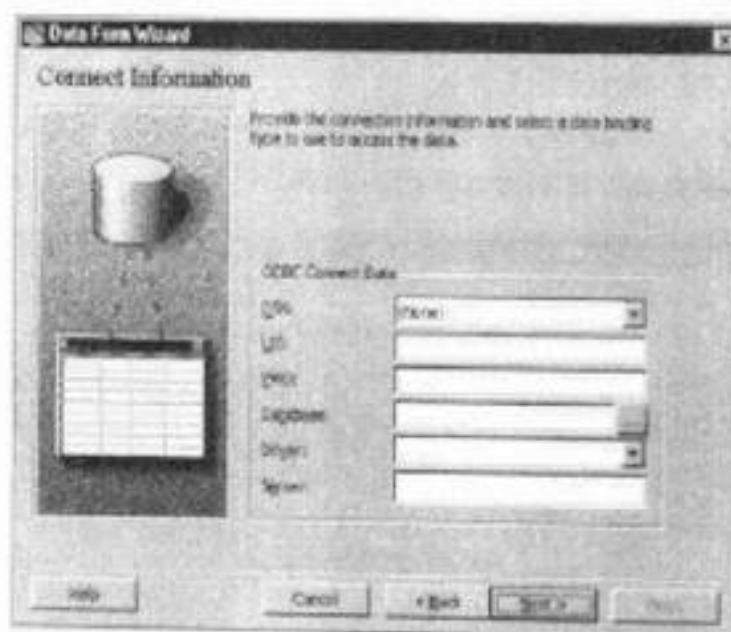


图 11.9 Data Form Wizard 的 Connect Information 步骤

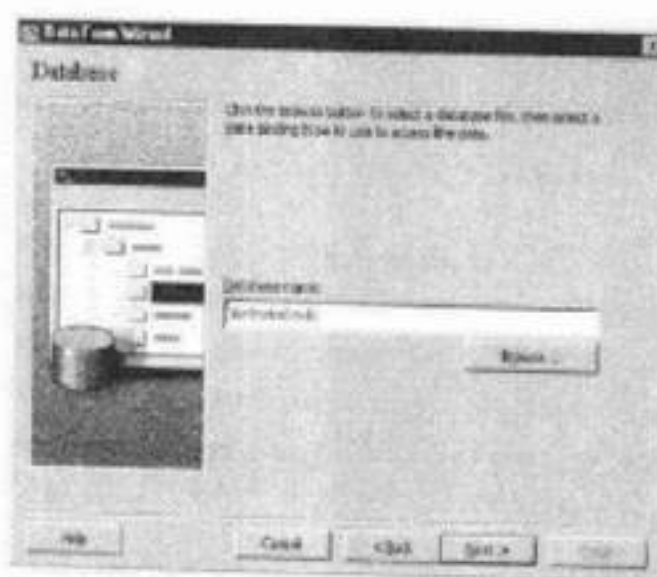


图 11.10 Data Form Wizard 的 Database 步骤

11.5.3 指定窗体类型

接下来向导允许您指定要创建的窗体类型。Data Form Wizard 提供 3 种窗体：Single Record、Grid (Datasheet) 和 Master/Detail。图 11.11 显示了 Data Form Wizard 的 Form Type 步骤时的外观。

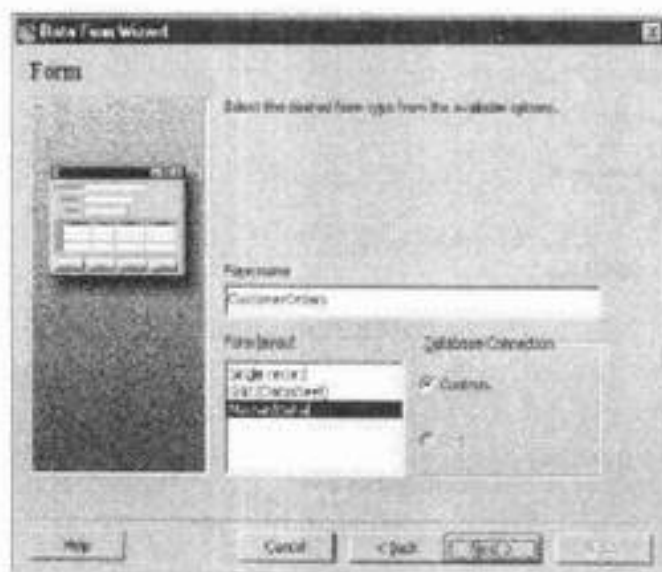


图 11.11 Data Form Wizard 的 Form Type 步骤

Single Record 窗体类型包含从数据库表中选择的每个字段的控件。Data Form Wizard 根据字段的数据类型选择控件。例如, Boolean 字段需要一个 Check-Box 控件; String 对象需要 Edit 控件。如果要使用户能够快速方便地一次访问多个记录, 使用这种窗体是很有用的。Master/Detail 窗体是一些显示多个表(一个字段中的记录在不同的表中有多个相关记录)的记录窗体, 我们将使用这种窗体来显示具有多个客户订单的客户信息。这种窗体是 Single Record 和 Grid (Datasheet)窗体的结合。我们从列表中选中 Master/Detail 选项。

在这一步中, 还可以更改在启动 Data Form Wizard 之前指定的窗体名称。并且, 如果选择的是 Single Record 或 Grid (Datasheet)窗体类型, 那么可以选择是使用代码还是使用控件来处理数据连接和绑定。通常情况下, 您会希望使用控件(对于 Master/Detail 窗体, 这是唯一的选项), 除非您想修改和添加向导提供的窗体。当熟悉 ADO 后, 代码选项将非常有用, 因为您可能希望更多控制连接和操纵数据库的方式。在这一步做出选择后, 就可以单击 Next 按钮, 进入下一步。

11.5.4 选择主要的和详细的记录源及字段

在上一步中, 我们使用 Data Form Wizard 来创建 Master/Detail 类型的窗体。在这一步中, 我们将设置 Master 记录源和 Detail 记录源。图 11.12 显示了 Data Form Wizard 的 Master Record Source 步骤。

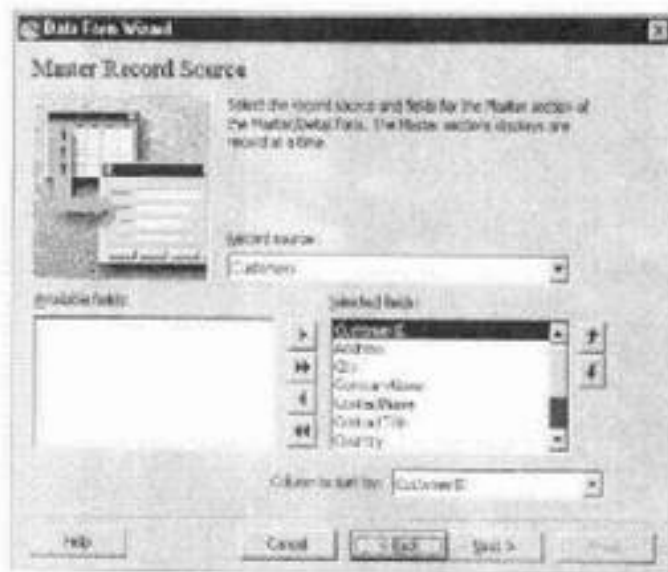


图 11.12 Data Form Wizard 的 Master Record Source 步骤

记录源是一个表。Master 记录源由表及 Master/Detail 窗体组成,用来显示 Master 记录的指定字段。Master 部分一次只显示一个记录。Detail 记录源是与 Master 记录相关的表。Master/Detail 窗体的 Detail 部分是显示 Detail 记录源记录指定字段的 DataGrid 控件。下面我们开始配置 Master 记录源。从 Record Source 下拉列表中选择 Customers 表,Available Fields 列表将使用表的字段名称来填写。单击双箭头按钮,将所有的字段添加到 Selected Fields 列表中。添加完字段,需要设置它们的顺序,以便 Data Form Wizard 添加到窗体中的控件按照更合理的顺序排列。通过选定字段名,然后使用列表旁的按钮向上或向下移动选定的字段。按照下面的方式设置字段顺序:

1. CustomerID
2. ContactName
3. ContactTitle
4. CompanyName
5. Address
6. City
7. Region
8. Country
9. PostalCode
10. Phone
11. Fax

现在可以从 Column To Sort by 下拉列表的字段列表选定一个字段,记录集

将根据该字段来排序。在我们的实例中,将根据 CustomerID 排序。设置完顺序后,单击 Next 按钮,进入到 Detail Record Source 步骤。它将使用 Northwind 的 Orders 表。选定 Orders 表,将所有的字段(除了 CustomerID 之外)添加到 Selected Fields 列表中。将 Column To Sort By 下拉列表设置为 Order Date 字段。像 Master 记录源那样,Detail 记录源需要更合理地组织。按照下面的方式排列 Selected Fields 列表中的字段:

1. OrderID
2. OrderDate
3. Freight
4. EmployeeID
5. RequiredDate
6. ShipName
7. ShipAddress
8. ShipCity
9. ShipRegion
10. ShipCountry
11. ShipPostalCode
12. ShippedDate
13. ShipVia

单击 Next 按钮,进入向导的下一步。

11.5.5 选择记录源关系及控件

Data Form Wizard 的这一步完成 Master/Detail 窗体的设置,并允许我们定义 Master 与 Detail 记录源之间的公共字段。图 11.13 显示了该步向导的外观。

显示两个列表:一个具有 Master 记录源的字段;一个具有 Detail 记录源的字段。Customers 和 Orders 表共享公共的 CustomerID 字段。每一个客户都有一个唯一的在 Customers 表中定义的 CustomerID,但是 Orders 表可以存在多个具有相同 CustomerID 的记录。这反映了两个表之间的一对多关系,并且正是这种关系使 Master/Detail 窗体能够正常工作。在两个表中选定 CustomerID 字段,然后单击 Next 按钮,继续向导的 Control Selection 步骤。

Control Selection 步骤允许您定义协助控制数据库记录的按钮。这一步还确定是否显示 DataNavigator 控件。图 11.14 显示了向导的 Control Selection 步骤。

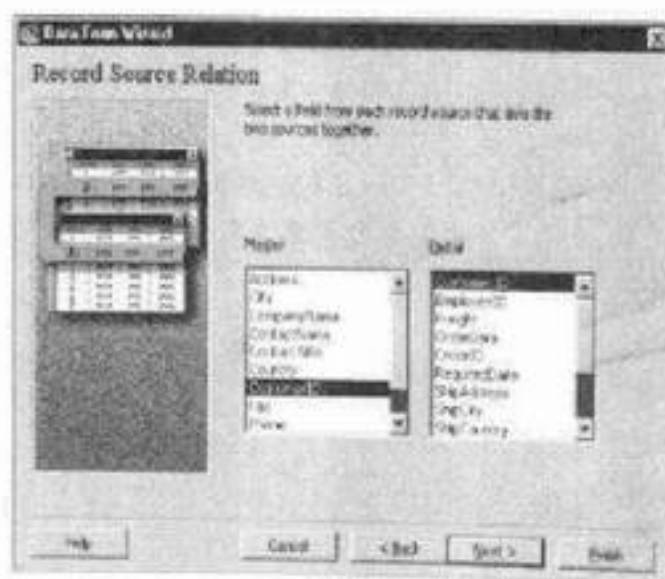


图 11.13 Data Form Wizard 的 Record Source Relation 步骤

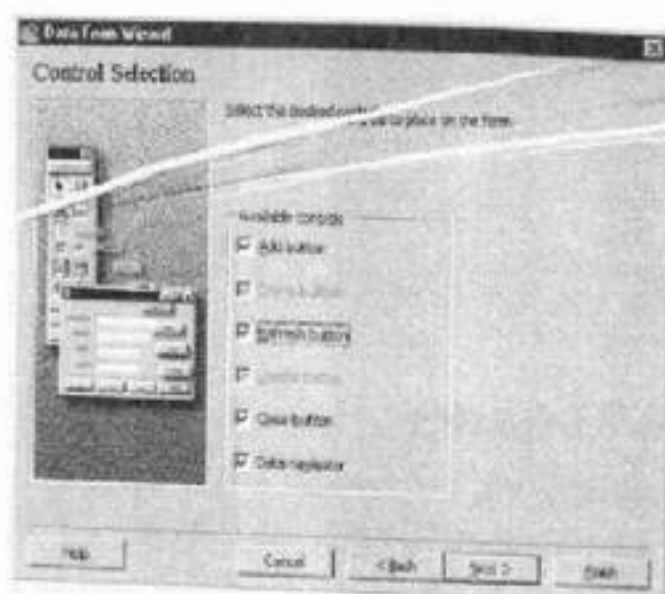


图 11.14 Data Form Wizard 的 Control Selection 步骤

DataNavigator 控件可以定位记录集中的记录。您可以从 Add、Delete、Refresh、Update 及 Close 按钮中进行选择。甚至可以选择 Data Navigator 控件。除了 Close 按钮,其他按钮都在数据库中执行一种操作。当您首先输入向导时,一些选项将被灰化,因为在选定 Add 按钮时,代码要求将这些按钮添加到对话框中。使用对话框默认的选项,显示所有的按钮,然后单击 Next 按钮,继续 Data Form Wizard 的最后一步。

11.5.6 完成向导,检查窗体并运行实例

最后一步把在向导中所做的设置保存到文件中,以便在 Data Form Wizard 后面的对话中重新使用它们。如果您计划创建多个相关的窗体,那么保存将是很有用的。图 11.15 显示了向导的最后步骤。

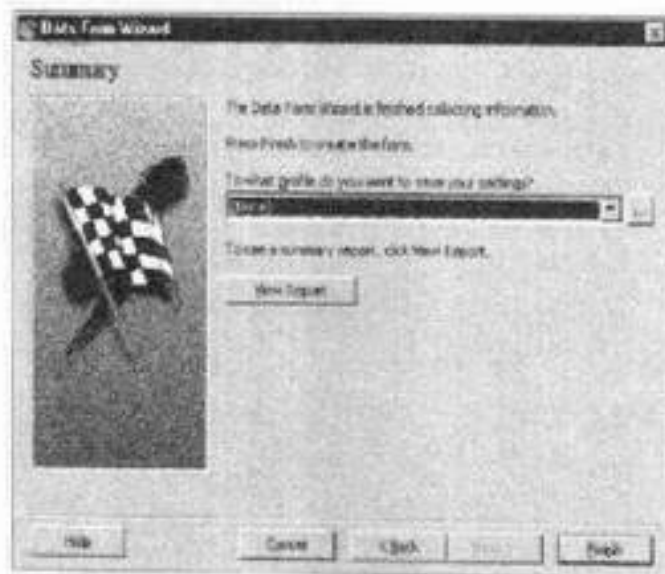


图 11.15 Data Form Wizard 的 Summary 步骤

单击 Finish 按钮,向导就构建我们设计的窗体。图 11.16 显示了 Data Form Wizard 为我们创建的窗体。

正如您所看到的,这个窗体非常大。建议您调整控件大小并精简窗体,因为使用不同分辨率的用户可能会遇到定位窗体的困难。在调整控件大小时,您或许会注意到控件都是固定的。Data Form Wizard 添加固定点可以更容易地处理各种分辨率。您还会注意到向导添加了 DataSource 控件和 DataBinder 控件。CustomerOrders 窗体使用 DataSource 控件连接数据库,并创建和存储 DataBinder 控件使用的记录集。DataBinder 控件存储窗体中所有的数据绑定连接。Data Form Wizard 使用该控件将数据库中的数据绑定到它添加的控件中。

编译并运行应用程序。当窗体显示时,注意它在顶端的控件中显示的客户记录,以及 DataGrid 控件显示的客户订单信息。更改其中一个字段 (Master 或 Detail 部分) 的值,然后单击对话框底部 DataNavigator 控件上的向右箭头按钮,记录将更改为新的记录。通过单击 DataNavigator 控件上的向左箭头按钮,返回到您更改的记录,该记录将重新显示刚刚应用的更改,这是数据绑定的作用。控件和基本记录集是绑定的,更改其中一个将可能弹出消息框或导致其他存储数据

的变化。试着使用窗体中不同的按钮,了解它们进行的操作。

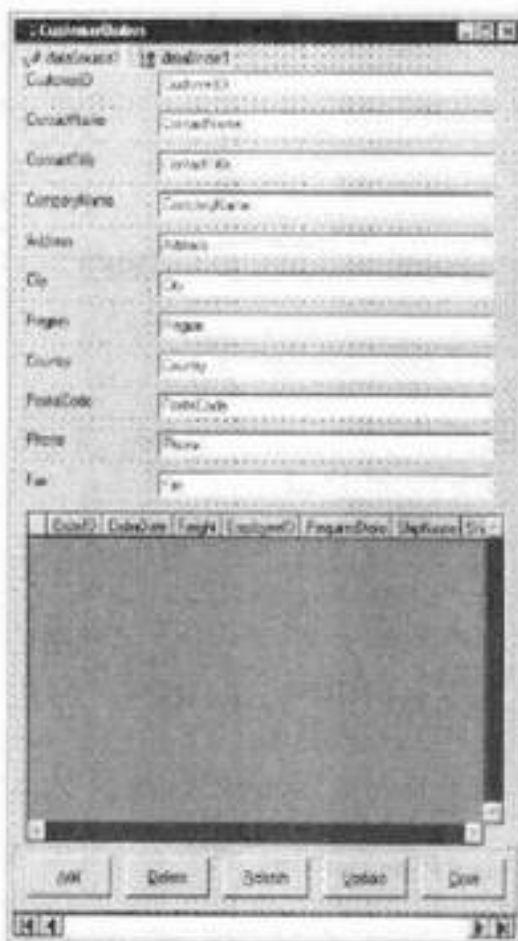


图 11.16 设计时的 CustomerOrders 窗体

正如您所看到的,Data Form Wizard 提供了一个略做修改就可以在实际应用程序中使用的窗体。如果需要,您可以看一看提供的代码。还可以想出一些方法来更改该实例窗体,以使它更为强大,但在这里,我们没有谈到如何操纵控件的绑定,或者如何通过代码与 ADO 进行交互。阅读完本节之后,您可以自己尝试一下。



更上一层楼

本章我们学习了有关数据库编程的一些内容,了解了 Visual J++ 提供的数据库对象——ADO,学习 ADO 的数据绑定模式,并用 ADO 创建了简单而实用的应用程序。学习了使用 DataGrid 等数据控件,以及如何通过代码直接访问数据

库。最后,还学习了数据库应用程序的向导——Data Form Wizard。

其实数据库应用程序在实际应用中是十分复杂的。大一点儿的应用程序用到的表和数据库通常有很多个,数据类型多种多样,如何操作庞大的数据库而不出现错误对数据库应用程序永远是一个挑战。只有在不断的实践中才能熟练地掌握数据库的应用。进一步的学习可以考虑在现有程序基础上增加索引、分类查询、数据表的主副表联接查询等功能以及练习使用其他数据控件,还可以学习数据库在网络方面的应用,这方面可以参考有关 ASP、PHP 的书籍。向实用方向发展,还必须学习 Java 与大型主流数据库的结合使用,例如 SQL Server、Oracle、DB2 等。

第 12 章 WFC 控件开发

知识要点:

- ◆ 子类化控件
- ◆ 自定义控件
- ◆ 使用 WFC Component Builder 和 Class Outline
- ◆ 使用 UserControl 类及其他 WFC 控件来创建组合控件
- ◆ 如何为控件中的属性创建自定义值编辑器
- ◆ WFC 到 ActiveX 的转换

在前面的章节中,我们了解了 Visual J++ 的开发环境和 Java 语言的概况,并利用 Visual J++ 开发环境进行了基本的 Java 编程,包括 Java 小程序和 Java 应用程序的编程。从本章开始,我们将介绍一些关于 Java 的高级主题:WFC 控件开发和 COM 组件开发。掌握这两个知识点,会使您在 Java 编程方面发挥更大的创造性,因为您将不仅会应用 Java 类库提供的大多数控件和组件,而且还会自己开发新的控件和组件。最后通过一个 Java 应用程序的高级实例来综合运用前面所有的知识。

在本章中,我们将介绍 WFC 组件模型,以及创建现有 WFC 控件子类的控件,在新建控件的基础上添加一些附加功能、属性和事件。最后,学习将 WFC 控件导出为 ActiveX 控件。通过使用本章中的工程,您将理解创建控件的过程,并且可以使用 Visual J++ 提供的工具来简化这一过程。



光盘 参阅本书配套光盘中的【Java 中的软件包】,可交互学习与本章内容相关的知识。

12.1 子类化控件

通过前面的学习,您应该对 Microsoft Visual J++、WFC 及 WFC 组件模型提供的功能有一个较好的理解。Visual J++ 与 WFC 控件一起附带的 Forms Designer 使您可以开发几乎能够满足各种用户界面要求的对话框和窗口。这种能力非常好,但是在 Visual J++ 开发应用程序的过程中,您也许希望创建自己的控件来满足特有的用户界面需求。您也许还会遇到这种情况,Visual J++ 附带的(或第三方开发商提供的)WFC 控件提供了您需要的大多数功能,但却缺少您的应用程序所需的某个重要功能。

Visual J++ 和 WFC 使您可以方便地创建自定义的 WFC 控件。无论它是其他控件的组合、现有 WFC 控件的扩展,还是全新的手工创建的控件,Visual J++ 都提供了一些工具和功能,使创建自定义控件更加容易。WFC 组件模型允许您开发能够在 WFC 应用程序中使用的强大的小型控件。

除了创建自定义 WFC 控件之外,您还可以将它们导出为 ActiveX 控件,然后在 Visual Basic 和 Office 等环境下使用它们。这种功能使在 Visual J++ 中进行控件开发更具吸引力。ActiveX 是一个令人发怵的词,它是具有大量功能的复杂结构,通常情况下,简单的控件不使用它。使用 WFC 控件模型,可以方便快速地创建组件。无论是创建复杂的、自己绘制的控件,还是创建现有控件的组合,您都会发现 WFC 控件模型相当直观和易于使用。

在本章中,我们将研究各种可以创建的自定义 WFC 控件类型。我们将忽略现有的 WFC 控件,来添加处理特殊用户界面需要的自定义功能。我们还将从头创建一些处理它们自己的图形显示的控件。理解如何构建标准的自定义控件后,我们将创建组合控件:由其他控件组成的控件。完成本章后,我们将能够在其他支持 ActiveX 控件的应用程序中使用的 ActiveX 控件来展示您定制的 WFC 控件。您将能够使用 Visual J++ 和 WFC 来创建一些满足应用程序编程需要的控件。

您可能多次遇到这种情况:有一个很棒的要用于应用程序的控件,例如一个日历控件,但是它没有提供应用程序所需的功能。这种情况对于很多使用 ActiveX 控件的开发人员来说是很常见的。如果幸运的话,可以从第三方获得一些提供了所需功能的 ActiveX 控件,那么它们或许可以将一些功能添加到控件中(如果它们有源代码)。但是具有源代码的控件是很少的,因此您无法改动它们以使它们充分满足需要。

即使有 ActiveX 控件的源代码,仍要做大量的工作。ActiveX 技术比较复杂,

并且对于普通的开发人员,它也比较难于使用。如果使用 Visual C++ 和 WFC (Microsoft 基类)开发 ActiveX 控件,或许可以将控件子类化,以添加一些小的功能。如果正在使用 Visual Basic,那么除了访问源代码之外,您别无选择,因为 Visual Basic 目前不支持继承性。对于使用 Visual Basic 开发的 ActiveX 控件,修改源代码并不是很难,虽然您必须重新构建整个控件以添加附加的功能。

WFC 组件模型包括最好的 ActiveX。像 Visual Basic 和 Visual C++ 一样,WFC 组件模型提供了一些面向对象,能够子类化,并且易于构建的控件。如果有未提供所需功能的 WFC 控件,那么您只需在新的源文件中将该控件子类化,然后添加所需的功能。正如您将要在我们打算开发的子类控件实例中看到的,WFC 控件模型使完成这个过程变得非常容易!

在该实例中,我们构建一个 NumTextActiveX 控件的副本,该 ActiveX 控件是一个限制条目数的 Edit 控件。要在 Visual Basic 中开发 ActiveX,我们不得不使用 UserControl(要了解 UserControl 的详细信息,请参见 12.3 节“组合控件”的有关内容),并在 UserControl 上放置一个 Visual Basic 的 Text 控件。

在 WFC 控件版本中,我们将简单地把 Edit 控件子类化,然后使用 Visual J++ 的 WFC Component Builder(WFC 组件生成器)来提供我们自己的功能,其中包括自定义属性和事件。该事件是 ActiveX 没有提供的功能。我们还将使用 Class Outline 来忽略 Edit 控件的一些属性,以便更好地处理控件对数字条目的需要。完成该实例后,不仅能够很好地将 WFC 控件子类化,以提供增强的功能,而且还能够使用 WFC Component Builder 来添加属性和事件,以及使用 Class Outline 来忽略属性方法。

12.1.1 创建控件工程

Visual J++ 提供了一个 Control 模板(位于 New Project 对话框的 Components 节点),您可以用来制作用户控件(要了解使用该模板的详细信息,请参见本章关于创建组合控件的内容)。因为 NumText 控件没有使用 Control 模板,所以我们将使用一个空工程,然后添加我们的类文件。使用 Empty Project 模板创建一个空工程,并将其命名为“NumText”。使用 Add Item 对话框中的 Class 模板将类文件添加到工程中,并将其命名为“NumText.java”。该文件将作为控件的类。因为 Class 模板提供了一些简单的类,所以我们需要为它添加一些信息。将下面的导入语句添加到类文件的上面:

```
import com.ms.wfc.core.*;  
import com.ms.wfc.ui.*;  
import com.ms.wfc.util.*;
```

这些语句使我们能够访问创建类,特别是要子类化的 Edit 类时所需的核心类。要将 Edit 控件子类化,我们还需要在类的名称后添加 `extends Edit`。扩展控件后,我们将获得原来 Edit 控件所有的默认属性和方法。如果需要,我们可以忽略这些方法。

12.1.2 创建 ClassInfo 类及 Value 属性

为了将属性和事件展示给控件的用户,Visual J++ 使用了我们所说的 ClassInfo 类:控件的嵌套类,其中所有属性和事件都是通过分别声明 PropertyInfo 和 EventInfo 类来定义的。除了定义属性和事件之外,ClassInfo 类还定义控件默认的属性和事件: `getEvents` 和 `getProperties` 等。`getEvents` 和 `getProperties` 方法使访问该控件的开发工具能够获得控件的属性和事件列表。

类似于 Forms Designer 管理窗体 `initForm` 方法的方式,WFC Component Builder 处理 ClassInfo 中的定义。它为属性和事件建立声明,并在 ClassInfo 类中为您提供所有必要的代码。WFC Component Builder 还为属性创建 `get` 和 `set` 方法,为事件声明创建 `add < EventName >`、`remove < EventName >` 及 `on < EventName >` 方法(这里 EventName 是事件的名称)。该工具消除了对理解在 ClassInfo 类中定义属性及事件所有细微差别的要求。

要访问 WFC Component Builder,必须在视图中打开 Class Outline 窗口。从 View 菜单中选择 Other Windows,然后选择 Document Outline,显示 Class Outline 窗口。通过右键单击 Class Outline 中控件类的名称,然后单击 WFC Component Builder 工程,来启动 WFC Component Builder。WFC Component Builder 将首先确定 ClassInfo 类是否在控件类中得到定义。如果没有,那么它将提示您创建一个。对于我们的控件,继续下面的操作,让它创建一个 ClassInfo 类。注意,WFC Component Builder 将 ClassInfo 类添加到控件的类文件中,并为 Delegate 类添加了一个附加的导入语句。您需要用这个语句来定义控件中的事件。

WFC Component Builder 还将一些定义添加到 `getEvents` 和 `getProperties` 的 ClassInfo 类中。一旦定义完属性和事件,WFC Component Builder 就将操纵这些方法。WFC Component Builder 在查看是否忽略控件类方面还相当智能。如果您忽略了控件类,那么它将把基控件的 ClassInfo 类子类化。默认情况下,这个过程将继承所有基控件的属性,它使将功能添加到现有控件中变得相当容易。

WFC Component Builder 创建完 ClassInfo 类,并将它添加到控件中后,您将看到一个类似图 12.1 所示的对话框。它允许您创建属性和事件,如果您已经有定义的属性和事件,那么它允许您删除它们。下面我们来创建一个新的属性 value。

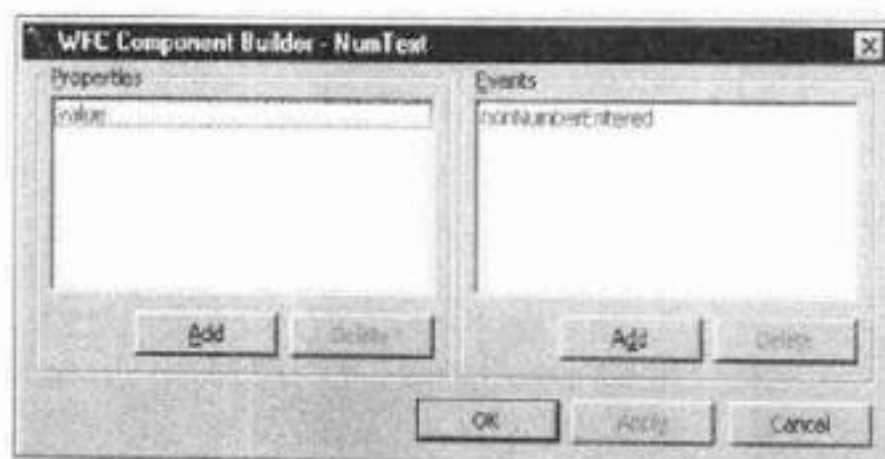


图 12.1 WFC Component Builder

value 属性将是我们展示控件中文本的数字版的途径。单击对话框 Properties 区中的 Add 按钮,显示 Add WFC Property 对话框。这里可以定义控件的名称、控件的数据类型(可以是任何有效的数据类型或对象类型)、属性的类别(在显示 Properties 窗口的 Categorized 视图时使用),以及属性的描述(显示在 Properties 窗口较低的区域中)。还可以将属性定义为只读(这意味着它将只创建属性的 get 方法),确定控件是否使用成员变量来存储它的状态,以及在使用时修改成员变量的名称。使用表 12.1 来定义 value 的属性。

表 12.1 value 属性定义

添加的 WFC 属性字段	设置
Name	value
Data Type	long
Category	Data
Description	Returns a numerical version of the text in the control
Read Only	checked
Declare Member Variable	unchecked

输入属性信息后,单击 Add 按钮,将它添加到要定义的属性列表中。单击 WFC Component Builder 中的 OK 按钮。WFC Component Builder 将为我们的 value 属性添加 PropertyInfo 定义,并为我们添加 getValue 方法来实现属性。我们将在稍后的实例中做这些工作。

12.1.3 定义 NonNumberEntered 事件

控件开发的另一个复杂领域是定义事件。除了 EventInfo 定义之外,事件需要 add 和 remove 方法来使控件的用户能够添加和删除事件的处理程序,并且它们需要一个实现事件指定代码的 on 方法。如果事件需要指定类型的代理和事件对象,那么还需要在控件中定义这些。虽然事件的代码不是很复杂,也不是很长,但是如果不能正确地完成,那么将产生问题。WFC Component Builder 可以使创建事件像创建属性一样简单。

我们创建的事件被称为“nonNumberEntered”。当用户将非数字的键盘字符输入到控件中时,将触发该事件。事件将使用一个被传递到事件处理程序(如果用户定义了它)中的自定义事件对象 NonNumberEvent,来提供击键信息,并且该事件将使用一个名为“NonNumberEventHandler”的自定义代理。在可以使用 WFC Component Builder 来添加事件的定义之前,我们需要定义 NonNumberEvent 和 NonNumberEventHandler。要定义它们,可以将两个类文件添加到工程中,然后将它们分别命名为“NonNumberEvent.java”和“NonNumberEventHandler.java”。在 NonNumberEvent.java 文件中,替换成如下所示的代码:



```
import com.ms.wfc.core.*;
public class NonNumberEvent extends Event
{
    public String keyPressed;

    public NonNumberEvent(String keyPressed)
    {
        this.keyPressed = keyPressed;
    }
}
```

我们添加的代码定义了从 Event 获得的对象。它展示了一个存储用户输入到控件中非数字击键信息的简单字段。现在将下面的代码添加到 NonNumberEventHandler.java 中来替换现有的代码:

```
import com.ms.wfc.core.*;
public final multicast delegate
    void NonNumberEventHandler(Object source, NonNumberEvent e);
```

该代码声明了一个将 NonNumberEvent 的实例作为参数接受的代理 NonNumberEventHandler。该代理与事件源相连接,在我们的 nonNumberEntered 事件中,代理与用户的事件处理程序相连接。

下面我们来添加事件。打开 NumText 的代码窗口,显示 NumText 类的 WFC Component Builder。在 WFC Component Builder 中,单击 WFC Component Builder 的 Events 区中的 Add 按钮,显示 Add WFC Event 对话框。Add WFC Event 对话框中的字段允许您定义事件的名称、创建的事件类型(Event 类或其子类)、事件的类别以及描述等。使用表 12.2 制定创建 nonNumberEntered 事件的信息。

表 12.2 nonNumberEntered 事件定义

添加的 WFC 事件字段	设置
Name	nonNumberEntered
Type	NonNumberEvent
Category	Key
Description	Occurs when a non-numerical key is pressed in the control

因为我們是在创建事件之前创建 NonNumberEventHandler 代理的,所以我们可以从列表中选择事件类型或者自己键入,因为它是存在的,所以我们可以将它连接起来。如果事先没有创建代理,那么您将不得不从列表中选择,稍后在创建代理和事件对象时在代码中更改它。稍后在代码中必须做的更改可能会导致错误。

单击对话框中的 Add,然后单击 WFC Component Builder 中的 OK 按钮,将事件代码添加到控件中。WFC Component Builder 将把代理 NonNumberEventHandler 的实例添加到新代码的上面,然后添加 add、remove 和 on 方法,以及事件的 EventInfo 定义。还需要在 getEvents 方法中添加一行代码,来将事件添加到控件的事件列表中。

12.1.4 将代码添加到 GetValue 方法中

使用 WFC Component Builder 创建事件和属性是相当容易的!下面我们来看一看它为我们的 value 属性提供的内容,然后添加一些代码来处理控件中文本数字化值的请求。将下面的代码添加到 getValue 方法中:



```
long returnValue = -1;
if (this.getText().equals("") == false)
{
    try
    {
        returnValue = Value.toLong(this.getText());
    }
    catch (Exception e)
    {
    }
}
```

```

        MessageBox.show("Numerical value too large for long"
            + "data type. Decrease"
            + "maxLength property",
            "NumText Control",
            MessageBox.ICONSTOP);
        returnValue = -1;
    }
    return returnValue;
}

```

我们添加到 `getValue` 方法中的代码定义将要存储在控件中文本数字化值的长整型值。在 `try/catch` 块中,代码将试着将控件 `text` 属性中的文本数字化值分配给 `returnValue` 变量。它通过使用 `Value` 类的 `toLong` 方法将 `String` 值转换为长整型数据来进行这种数字化转换。如果该值对于长整型数据太大,那么将会发生意外情况,并会被捕获语句捕获。捕获块的代码只显示一个消息框,建议用户通过减小 `maxLength` 属性值来限制控件中可以显示的字符数。然后它将返回值设置为 `-1`,表示出现了错误。

我们可以根据这个错误来清除控件中的文本,但是最好不要更改输入的内容。用户也许想获得负值条件,然后在没有使用控件的数字化值时提取文本。

12.1.5 使用 Class Outline 忽略方法

为了确定用户按键或设置控件 `text` 属性的时间,我们需要忽略基类的执行代码,并提供我们自己的执行代码。忽略继承方法的一个很好的工具是 `Class Outline`。它允许您浏览指定类的继承方法列表,并忽略它们。要忽略 `Class Outline` 的 `Inherited Members` 节点的给定方法,只需右键单击方法名称,然后单击 `Override Method`。`Class Outline` 将您选定要忽略方法的一个方法模板添加到类的下面。

对于 `NumText` 控件,我们需要忽略 `setText`、`onKeyPress` 和 `onKeyDown` 方法。我们忽略 `setText` 方法是为了防止用户在 `Properties` 窗口或通过代码输入非数字化的字符。`onKeyPress` 和 `onKeyDown` 方法分别是 `keyPress` 和 `keyDown` 事件默认的处理程序。我们将使用这些方法来筛选出非数字化字符,并在输入非数字化字符时触发 `nonNumberEntered` 事件。下面我们来添加 `setText` 方法代码,替换该方法中现有的代码。



```

try
{
    long tempLong = Value.toLong(value);
}

```

```

|
catch (Exception e)
|
    value = "";
|
super.setText(value);

```

该代码类似于我们为 `getValue` 方法添加的代码。它从为分配到控件 `text` 属性的文本值的数字化转换设置一个长整型变量开始。这种转换是在一个 `try/catch` 块中完成的。如果出现意外情况,那么所有文本的参数值将被清除。最后代码调用 `setText` 带有 `value` 参数的父类版本。该代码还有助于我们防止首先将控件添加到窗体时在控件中显示其名称。

在添加 `onKeyDown` 和 `onKeyPress` 方法的代码之前,我们需要添加一个私有成员变量,我们将在控件检测到非数字化字符时用来做标记。将下面的变量定义添加到类定义的前面:

```
private boolean m_bNonNumber = false;
```

我们将用以下的代码来替换 `onKeyDown` 方法中的代码:



```

// Determine if control entered is in range of numbers
if (e.getKeyCode() < 48 || e.getKeyCode() > 57)
|
// Determine if the keypad's keys are pressed
if ( e.getKeyCode() < 95 || e.getKeyCode() > 105)
|
// Determine if it is the backspace key
if (e.getKeyCode() != 8)
|
    // The key pressed is a non-number
    m_bNonNumber = true;
|
|
else
    m_bNonNumber = false;
|
else
// The key pressed is a number
m_bNonNumber = false;

// To ensure proper handling, call the super's version
super.onKeyDown(e);

```

onKeyDown 方法的代码通过检查输入到控件中的字符的 keyCode 值范围,来确定是否输入了非数字化字符。keyCode 是一个表示击键值的数字化值。键盘上面的所有数字的范围为 48 ~ 57,小键盘上的数字范围为 95 ~ 105。代码确定当前击键的 keyCode 是否是键盘上的一个数字。如果不是,那么代码将查看它是否是小键盘上的数字。如果都不是,那么代码将查看当前的 keyCode 是否为 8 (退格键)。如果是数字或退格键,那么 m_bNonNumber 将被设置为 false;否则,它将被设置为 true。onKeyPress 方法(在 onKeyDown 之后调用)将使用这个标记来继续处理击键。最后,代码将调用方法的父类,以使父类执行标准的方法处理。

最后我们来添加 onKeyPress 方法的代码。使用下面的代码替换 onKeyPress 方法中的代码:



```
// Determine if control entered is in range of numbers
if (m_bNonNumber)
{
    /* Consume the keystroke that was pressed
     * since it is not a number */
    e.handled = true;
    // Invoke the event handler for a non-number entry
    onNonNumberEntered(new NonNumberEvent(
        String.valueOf(e.keyChar)));
}
else
    super.onKeyPress(e);
```

当发生 keyPress 事件时执行 onKeyPress 的代码,并且正是在这个时候,击键的字符值已经得到确定和处理。代码从查看 m_bNonNumber 变量是否被设置为 true 开始。如果是,那么代码将把 KeyEvent 对象的 handled 字段设置为 true。这将告诉 WFC 该事件已经被处理,WFC 不需要再对事件做任何处理。

代码还将使用通过击键字符值的 String 值定义的 NonNumberEvent 事件对象实例来调用 onNonNumberEntered 方法。使用 KeyEvent 的 keyChar 变量获得字符。因为该变量属于 char 类型,所以 String 类的 valueOf 方法用于将 char 转换为 String。这种 m_bNonNumberEntered 调用是控件触发控件中的事件的方法。如果 m_bNonNumber 为 false,那么将调用该方法的父类版,以确保正确处理方法。

在编译和生成控件之前我们需要执行的最后一项任务是:为调用超级构造函数的控件添加一个构造函数。该构造函数使 NumText 控件能够从所有 Edit 类的初始化过程中获得益处。



```
public NumText()
{
    super();
}
```

添加完构造函数后,构建控件,然后将其添加到 Toolbox 中。从 Tools 菜单中选择 Customize Toolbox。这时,NumText 将显示在 Manage Toolbox 对话框 WFC Controls 选项卡中。选定 NumText,然后 Visual J++ 将把它添加到 Toolbox 中。因为我们还没有定义该控件,所以它有一个相关的 Visual J++ 图标(我们将在稍后的实例中演示如何做到这一点)。

12.1.6 将控件添加到窗体中

要测试控件,可以将一个窗体添加到工程中,并将其命名为“NumTextTest.java”。要使查看控件的实际效果更加容易,在本章中我们只需将一些窗体添加到控件所在的工程中,然后以一种简单的方式而不是完整的实例在窗体中测试它们,因为控件是该实例中最重要的部分。

将窗体添加到工程中后,单击 Toolbox 中的 NumText 控件,将其添加到窗体中。在将控件添加到窗体中时,还要添加一个按钮。我们将使用该按钮来调用 getValue 方法,以显示控件的数字化值。通过在 Events 视图中双击 click 事件,创建该按钮 click 事件的处理程序。也可以通过在 Properties 窗口双击事件来添加 NumText 控件 nonNumberEntered 事件的处理程序。

乍一看,控件除了显示一个空的 Edit 控件之外没有任何内容。试着在 Properties 窗口中将它的 text 属性设置为数字化值。该控件接受数字。现在将一些非数字字符添加到 text 属性中。按下 Enter 键,那么控件将转换为空设置,这是 setText 的效果。在设置属性过程中没有调用 onKeyPress 和 onKeyDown 方法的原因是只有在运行时使用控件,以及在用户直接输入字符时,才会调用它们。要测试输入的值,可以将下面的代码添加到 Button 控件的 click 事件处理程序中:

```
MessageBox.show("The value from getValue is" + numText1.getValue());
```

我们添加的代码将显示一个消息框,指出调用 getValue 的返回值。如果输入的数太大,那么还会得到 getValue 方法为大于长整型值的情况定义的消息。现在我们将下面的代码添加到 nonNumberEntered 事件处理程序中来测试事件处理程序:

```
MessageBox.show("The character that was entered was" + e.keyPressed);
```

与按钮的事件处理程序一样,该代码将显示一个消息框。这时代码通过将 NonNumberEntered 事件当前的 keyPressed 值传递到消息框中,来显示输入的字符。编译并运行工程。将一些数字字符输入到控件中,然后单击该按钮。这时将显示出控件的数字化值。现在输入非数字字符来触发 nonNumberEntered 事

件。NumText 控件的事件处理程序将显示它的消息框。

编译并运行窗体后,您或许想增强控件。以下是一些建议。

- ◆ 将 setValue 属性添加到控件中,并允许用户使用数字化的值来设置它。必须确保输入的值能够像我们在 setText 方法中所做的那样将该值转换为 long 型。
- ◆ 提供一个定义是否允许负值(-)的属性。有些人或许希望显示它,而有些人则不希望这样做。最后,使它更加灵活一些。对于其他一些数字字符,例如加号(+),您可以做到这一点。请记住,如果修改 NumText 控件以接受负值,那么还必须修改 NumText 的 getValue 方法返回错误条件的方式。

12.2 自定义控件

在开发应用程序的过程中,您或许会发现当前 WFC 控件库中没有并且使用组合控件也无法有效准确使用的控件。这时,需要考虑创建自定义控件:从头开发的控件和其他没有子类版的控件。创建自定义控件通常需要付出许多努力,因为您必须处理很多工作。

幸运的是,com.ms.wfc.ui.Control 类通过为常见属性和事件提供默认执行代码来帮助您做到这些。在 Control 类的帮助下,您仍需要添加大量的代码来处理在屏幕上绘制控件的图形工作。这通常需要您将 paint 事件子类化,并处理所有的控件绘制工作。虽然这个过程听起来比较困难,但是如果您理解了如何在 Windows 中使用 GDI(对于 WFC,使用 Graphics 类方法)进行绘制,那么它实际上并不是很困难。

在本节中,我们将设计一个不断向用户显示当前时钟的自定义控件。图 12.2 显示了它在运行时的情况。该控件将处理一些任务,例如响应 paint 事件,设置窗口样式,以及提供边框绘制支持等。完成该实例后,您将会理解开发自定义控件的步骤。

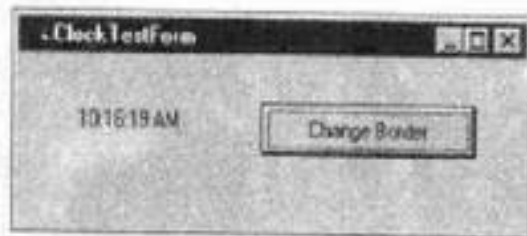


图 12.2 运行时的 Clock 控件

12.2.1 创建控件工程

像 NumText 控件一样, Clock 控件不使用 Visual J++ Control 模板, 而是使用 Empty Project 模板。使用 Empty Project 模板创建一个新的工程, 然后将其命名为“Clock”。将一个类添加到工程中, 并将其命名为“Clock.java”。将下面的导入语句添加到类文件的前面:



```
import com.ms.wfc.ui.*;
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.win32.*;
import com.ms.win32.*;
```

声明 Clock 类, 以便它扩展 Control 类。这将为控件提供默认属性和事件。

12.2.2 将代码添加到构造函数中

因为我们在没有像 Edit 控件等基本控件(事先定义的控件类型)的帮助下创建控件, 所以我们必须在控件的构造函数中指定我们自己的控件。我们还将使用构造函数来定义当用户双击 Toolbox 中的控件时, 将它添加到窗体中的默认大小。该时钟控件将实时显示小时、分钟和秒。我们将会在构造函数中添加声明, 来定义 onIdle 事件的处理程序。当操作没有处理任何事件或方法时, 将发生这个事件, 它会更新时钟的显示位置。将下面的构造函数添加到 Clock 类中:



```
super();
Application.addOnIdle(new EventHandler(DrawTime));
setStyle(STYLE_USERPAINT, true);
setStyle(STYLE_SELECTABLE, false);
setTabStop(false);
setSize(70, 15);
```

该构造函数的第一行代码调用 Control 类的构造函数, 它将为控件提供默认设置。接下来的代码将声明 onIdle 事件处理程序, 以便我们可以更新时钟的显示。然后代码初始化 Clock 控件的样式。STYLE_USERPAINT 样式指出控件自己进行绘制工作。因为控件处理所有的绘制工作, 所以这种样式非常重要。最后一个样式 STYLE_SELECTABLE 被设置为 false, 因为我们的控件在窗体中不是一个可选控件, 它是当前时间的静态显示。该构造函数的代码最后设置控件的默认大小, 并将 tabStop 属性设置为 false。将 tabStop 设置为 false, 是因为我们创建的不是可选控件。

在完成使用构造函数之前,我们将为构造函数中声明的 `onIdle` 事件添加处理程序。将下面的代码添加到 `Clock` 类中:



```
private void DrawTime(Object sender, Event e)
{
    this.onResize(Event.EMPTY);
}
```

`DrawTime` 事件处理程序使用 `onResize` 方法来调用调整大小的事件。在该实例后面部分,将忽略这个方法。我们将使用调整大小的事件来处理时钟显示的初始化工作,并使控件显示失效,以使 `onPaint` 方法能够绘制时钟。在稍后的实例中,我们将详细介绍这些内容。

12.2.3 添加属性和事件

`Clock` 控件的一个功能是更改它的边框样式,使控件支持边框、三维效果,或者没有边框。诸如 `Label` 等控件具有这种功能。因为是从 `Control` 类获得这种功能,所以我们需要在 `Clock` 控件中提供这个属性作为自定义属性。下面添加这个属性,并添加在更改边框样式属性时发生的事件。虽然用户通常不需要这个事件,但是它可以根据当前程序的条件更改边框样式在 `Clock` 控件类中显示 WFC Component Builder。使用表 12.3 定义 `borderStyle` 属性。

表 12.3 `borderStyle` 属性定义

添加的 WFC 属性字段	设置
Name	<code>borderStyle</code>
Data Type	<code>int</code>
Category	Appearance
Description	Determines the type of border to be displayed
Read Only	unchecked
Declare Member Variable	checked
Member Variable Name	<code>m_BorderStyle</code>

我们在 `Add WFC Property` 对话框的 `Data Type` 字段中输入的 `BorderStyle` 类是一个枚举类,其他 WFC 控件使用它来定义允许的边框样式。如果在属性的 `Data Type` 字段中指定了枚举类,那么 WFC 将在 `Properties` 窗口中显示枚举字段。这个 WFC Component Model 功能非常有用,因为它使提供属性预先设置的值变得很容易。虽然 WFC Component Builder 为属性提供了大多数所需的信息,但是我们必须为该属性定义一个默认值,使它在显示控件时能够有一个初始值。将一个

新的 DefaultValueAttribute 对象添加到 borderStyle 属性的 PropertyInfo 定义中,然后将 PropertyInfo 构造函数的第 3 个参数从 int.class 更改为 BorderStyle.class。结果 borderStyle 的声明应该如下面代码所示:



```
new PropertyInfo(Clock.class,
    "borderStyle",
    BorderStyle.class,
    CategoryAttribute.Appearance,
    new DefaultValueAttribute(
        new Integer(BorderStyle.NONE)),
    new DescriptionAttribute(
        "Determines the type of border"
        + "to be displayed.")
);
```

在上述代码中,我们已经添加了一个新的 DefaultValueAttribute 对象,它定义 BorderStyle.None 字段初始化一个新的 Integer 对象。这将使 borderStyle 属性默认为无边框显示。DefaultValueAttribute 是可以添加到属性定义中来添加功能并指定属性设置的众多属性之一。要了解关于如何分配给属性的不同属性对象的详细信息,请参见 Visual J++ 的联机文档。

将附加的代码添加到 setBorderStyle 方法中,如下所示:



```
if (! BorderStyle.valid(value))
    throw new WFCInvalidEnumException("value",
        value,
        BorderStyle.class);

if (m_BorderStyle != value)
{
    m_BorderStyle = value;
    recreateHandle();
    this.onBorderStyleChanged(Event.EMPTY);
}
```

代码首先通过调用具有分配值的 valid 方法来确定被分配的值是否有效。所有枚举类的 valid 方法确定被分配的值是否在类的有效值范围内。如果该值不是有效的,那么将发生一个意外情况。然后代码查看当前边框是否与分配的边框样式有所不同。如果不同,那么代码将设置 m_BorderStyle 的值来存储当前的边框样式。然后代码重新创建控件的处理程序,这将使控件把它的边框样式重新设置为当前定义的边框样式。最后,代码将调用带有空 Event 对象的 onBorderStyleChanged 方法来触发事件。

我们已经定义了属性,下面将把 borderStyleChanged 事件添加到控件中。与

前面实例中的 `nonNumberEntered` 事件一样,我们不必对 WFC Component Builder 提供的事件代码做任何更改。使用表 12.4 定义 `borderStyleChanged` 事件。

表 12.4 `borderStyleChanged` 事件定义

添加的 WFC 事件字段	设置
Name	<code>borderStyleChanged</code>
Type	Event
Category	Action
Description	Occurs when the <code>borderStyle</code> property changes

与我们在前面实例中添加的 `nonNumberEntered` 事件不同,这个事件不需要定义任何特殊的事件对象或代理。`borderStyleChanged` 事件使用了一个标准的 `Event` 对象,因为它没有将任何特殊信息提供给事件处理程序。

12.2.4 忽略 Control 类的方法

为了完成控件,需要忽略一些 `Control` 类的方法,以提供我们自己的执行代码。使用 Class Outline 忽略 `onResize`、`onPaint`、`onEraseBackground` 和 `getCreateParams`。在将代码添加到这些忽略的方法之前,我们需要添加一个成员变量,来协助绘制控件的时间显示方式。将下面的成员变量声明添加到 `Clock` 类的前面:

```
Bitmap tempBuf = null;
```

定义完成成员变量后,我们将把代码添加到我们忽略的方法中。



```
protected void onResize(Event e)
{
    // Determine whether the buffer is null
    if (tempBuf != null)
    {
        // Dispose the buffer...
        tempBuf.dispose();
        // ...and set it to null
        tempBuf = null;
    }

    // Create a point object with the size of the control
    Point tempSize = getClientSize();
    /* Initialize the buffer with a new bitmap
     * with the size of the control */
    tempBuf = new Bitmap(tempSize.x, tempSize.y);
    /* Invalidate the control's display
```

```

        * to force onPaint to be called */
        invalidate();
        // Call the super class implementation of the method
        super.onResize(e);
    }

    protected void onPaint(PaintEvent e)
    {
        // Create a Graphics object for the buffer
        Graphics g = tempBuf.getGraphics();
        // Get the client area dimension
        Rectangle clientArea = getClientRect();

        /* Set the background color of the buffer
        * to the control's background color */
        g.setBackColor(this.getBackColor());
        /* Set the text color of the buffer to
        * the current foreColor property */
        g.setTextColor(this.getForeColor());
        /* Set the font of the buffer to the control's font */
        g.setFont(this.getFont());
        // Clear the contents of the control
        g.clearRect(clientArea);
        /* Draw the current time centered horizontally
        * and vertically */
        g.drawString(new Time().formatLongTime(),
                     getClientRect(),
                     TextFormat.HORIZONTALCENTER
                     |TextFormat.VERTICALCENTER);
        // Dispose the graphics object to free resources
        g.dispose();

        // Draw the buffer onto the control's surface
        e.graphics.drawImage(tempBuf,0,0);
    }

    protected CreateParams getCreateParams()
    {
        // Create an instance of the CreateParams class
        CreateParams tempParams = super.getCreateParams();
        /* Set the class name of the control to STATIC since it is
        similar to a static label control */
        tempParams.className = "STATIC";
    }

```

```

        // Determine the current border style to display
        switch (m.BorderStyle)
        {
            // The border style is a single lined border
            case BorderStyle.FIXED_SINGLE:
                tempParams.style |= win.WS_BORDER;
                break;
            // The border style is a sunken 3-D box
            case BorderStyle.FIXED_3D:
                tempParams.style |= win.SS_SUNKEN;
                break;
        }
        // Return the parameters to WFC so the control can be created
        return tempParams;
    }

    protected void onEraseBackground(PaintEvent event)
    {
    }

```

下面我们来看一看 `getCreateParams` 方法的代码。当控件被创建用来获得定义控件窗口外观的创建参数时, WFC 将调用该方法。这些创建方法允许我们定义控件窗口的边框样式。`getCreateParams` 的代码从创建使用父类当前创建参数初始化的 `CreateParams` 对象开始。代码然后将控件的类名设置为 `STATIC`。该控件的类名是控件窗口的窗体类。`STATIC` 类名通知 Windows, 这个控件应该作为静态标签处理。

设置完控件的窗口类后, 代码将确定当前的边框样式, 并使用 Win32 数据包提供的边框样式常数来设置 `CreateParams` 对象的 `style` 字段。代码设置完窗口样式后, 它将把 `CreateParams` 对象返回给 WFC, 来创建具有定义的创建参数的控件。

现在我们已经定义了控件创建参数, 下面我们需要绘制时钟。要进行这项工作, 我们来看一看为 `onResize`、`onPaint` 和 `onEraseBackground` 方法添加的代码。这些方法组合用来执行实际的时钟显示操作。

首先来看一看 `onResize` 方法。回顾我们在讨论 `DrawTime` 方法时, 当每次触发 `idle` 事件时, 它都将调用 `onResize` 方法。因为控件显示时钟的秒, 所以它将每秒更新一次, 我们要防止它在显示时出现闪动情况, 并提供一个平滑的时钟绘制过程。要做到这一点, 我们将使用一种被称为“双缓冲”的技术, 它将缓冲绘制过程, 以使绘制例程不会直接在控件中完成。为了做到这一点, 我们使用前面在类中定义的 `Bitmap` 对象, 并绘制它来代替绘制控件本身。

`onResize` 的代码从确保 `tempBuf` `Bitmap` 对象为空开始。我们不需要使用位图以前的内容来执行我们新的时间显示操作。位图被清空后, 代码将 `tempBuf` 初

始化为带有控件客户区维数的新的 Bitmap 对象。然后代码调用控件的 `invalidate` 方法来使控件重新绘制。一旦代码使控件失效后,代码最后将调用 `onResize` 方法的父类执行代码。

一旦 `onResize` 方法使控件失效后, `paint` 事件将调用 `onPaint` 方法。`onPaint` 方法的代码以创建 `tempBuf` Bitmap 对象的 Graphics 对象开始。Graphics 对象使我们能够绘制 Bitmap 对象。然后代码继续设置控件的字体、文本颜色,以及 Graphics 对象的背景颜色。一旦 Graphics 对象准备使用控件当前图形属性进行绘制时,代码就调用具有控件维数的 `clearRect` 来清除 `tempBuf` 的内容。然后代码调用 Graphics 类的 `drawString` 方法来将当前时间绘制到 `tempBuf` Bitmap 对象中。另外,绘制时间将使用控件的当前图形属性设置。这使用户可以配置时间的显示方式。

为了获得当前时间,代码创建了一个没有参数的 `Time` 类实例。这将创建一个具有当前时间的 `Time` 对象。为了将 `Time` 对象转换成 `String` 对象,代码将使用 `Time` 类的 `formatLongTime` 方法。为了使时间总是显示在控件中央,将使用 `TextFormat` 类的 `HORIZONTALCENTER` 和 `VERTICALCENTER` 字段调用 `drawString`。通过传递控件的客户区, `drawString` 方法可以将文本在控件边框内居中。`drawString` 方法将当前时间绘制到 Bitmap 对象后,代码将展示 Graphics 对象。代码最后调用 `drawImage` 方法来将 `tempBuf` 绘制到控件的表面上。

我们忽略的最后一个方法是 `onEraseBackground` 方法。当控件在绘制前需要擦除背景时将使用该方法。因为擦除背景常常会产生闪动,所以我们只需删除父类 `onEraseBackground` 事件处理程序的调用。

现在就完成了控件的代码。构建控件并将它添加到 Toolbox 中。在下一步骤中,我们将把该控件添加到测试窗体中,以查看它的实际效果。

12.2.5 将 ClockTestForm 窗体添加到工程中

为了测试控件,我们需要将窗体添加到工程中,然后将控件添加到窗体中。添加新的窗体,并将其命名为“`ClockTestForm.java`”。将 `Clock` 控件添加到 Toolbox 中,然后将它添加到窗体中。还要将一个 `Button` 控件添加到窗体中,并将它的 `text` 属性设置为 `Change Borderstyle`。通过双击 Properties 窗口中的事件名称来创建 `Button` 的 `click` 事件的处理程序。另外,使用默认的事件名称创建 `Clock` 控件 `borderStyleChanged` 事件的处理程序。将 `Clock` 控件添加到窗体中后,您将会注意到它显示当前的时间,但没有变化。原因是控件在窗体运行之前没有获得 `idle` 事件。如果调整控件的大小,那么您要注意,时间总是显示在控件中央。另一个要注意的问题是当您打开 Properties 窗口中 `borderStyle` 属性旁的下拉列表时,将显示可用的边框样式列表。

将下面的代码添加到 Button 控件的 click 事件处理程序中:



```
if (clock1.getBorderStyle() == BorderStyle.NONE)
    clock1.setBorderStyle(BorderStyle.FIXED_3D);
else if (clock1.getBorderStyle() == BorderStyle.FIXED_3D)
    clock1.setBorderStyle(BorderStyle.FIXED_SINGLE);
else
    clock1.setBorderStyle(BorderStyle.NONE);
```

该代码只在控件所有可用的边框样式中循环,并在列表中设置下一种样式。这种循环使您可以通过单击按钮来将 borderStyle 属性更改为不同的边框样式。

将下面的代码添加到 borderStyleChanged 事件的处理程序中:

```
MessageBox.show("The border style has changed!");
```

该代码只调用使您了解 borderStyle 属性已经更改的消息框,编译并运行代码。当窗体显示时,您会注意到时钟本身在不断地更新,但没有任何闪动。单击 Button 控件,不仅会将 borderStyle 属性更改为不同的边框样式,而且还会显示消息框,通知我们边框已经更改。

现在我们已经创建完控件,您也许会对如何在应用程序中使用它有一些好的想法,您或许还考虑了一些改进它的方法。使用以下这些建议,以使控件的功能更加全面。

- ◆ 使控件不仅能够显示时间,而且还可以作为计时器。可以创建一个 style 属性,并让用户能够定义控件类型是标准时钟还是计时器。(定义一个枚举类,使这些类型可以从 Properties 窗口获得。)然后将控件用于时间的显示。当您需要了解客户在一个订单条目系统中与您通话的时间,或者一项任务完成的时间时,这可能会很用。
- ◆ 将提醒功能添加到控件中,以便在经过指定的时间时,能够提醒用户。

12.3 组合控件

Visual Basic 最早的一个功能是它能够创建 ActiveX 控件。它通过允许用户将控件添加到 UserControl 对象中来完成这个操作。您可以绘制 UserControl 对象,但是它主要的用途是创建由其他控件组成的控件。WFC 除了具有标准控件创建能力之外,还在它的 UserControl 类中提供了这种功能。通过 UserControl 类创建的控件,不同于我们所说的组合控件,它对于将其他 WFC 控件组合到一个控件中,来提供强大的用户界面元素来说是一种有意义的方法。

组合控件可以进行常规的对话框显示,并且可以在一个能够在多个应用程序重复使用的控件中包含它们。例如,您可以创建一个组合 TreeView、ComboBox 和 Button 控件的控件,来提供一个可重复使用的目录选择用户界面。该控件可能具有一些属性,允许的用户获得 TreeView 控件中当前选定目录中的所有文件列表。组合控件是快速创建能够在任意应用程序中重复使用的组件的好方法。因为除了类之外它们没有别的什么,所以您可以通过将它们子类化来扩展它们的可重用性,以添加更多的功能,达到事半功倍的效果。

在实例中,我们将创建一个提供姓名和地址用户界面的组合控件。在运行期间,用户可以输入他稍后可以访问的每个字段或作为地址标签的姓名和地址信息。图 12.3 显示了控件运行时在窗体中的外观。在该实例中,将了解如何使用 Forms Designer 来布置 UserControl 表面上的控件,以及如何创建属性自定义值的编辑器。完成该实例后,您将知道如何使用 WFC 来创建由其他控件组成的控件。



图 12.3 运行时的 AddressProvider 控件

12.3.1 使用控件模板创建控件工程

在前面的实例中,我们在创建控件工程时不得不使用 Empty Project 模板。在该实例中,我们将使用 Visual J++ 的 Control 模板。它提供了从 UserControl 类获得的控件。您可以在 Forms Designer 中显示该控件,然后将一些控件添加到该控件中。使用 Control 模板(位于 Visual J++ Projects 的 Components 节点下)创建

一个新工程,并将其命名为“AddressProvider”。然后将工程中的类文件重新命名为“AddressProvider.java”,并将源代码中所有的 Control1 实例重新命名为“AddressProvider”。

12.3.2 将控件添加到 UserControl 中

将控件类重新命名为“AddressProvider”后,在 Forms Designer 中显示该类。UserControl 将在 Forms Designer 中作为一个基本框显示,您可以将控件添加到其中。将控件添加到 UserControl 中与将控件添加到标准 WFC 窗体中的方式相同。下面我们将组成组合控件的控件添加到 UserControl 中。使用表 12.5 来设计窗体中的控件。

表 12.5 AddressProvider.java 组件和属性设置

组件类型	属性	设置
Label	Name	lblTitle
	Text	Title:
	TabIndex	0
ComboBox	Name	cmbTitle
	Text	""
	Style	DropDownlist
ComboBox (continued)	TabIndex	1
	Items	Mr.
		Mrs.
		Miss.
		Dr.
Label	Name	lblFirstName
	Text	First Name:
	TabIndex	2
Edit	Name	txtFirstName
	Text	""
	TabIndex	3
Label	Name	lblLastName
	Text	Last Name:
	TabIndex	4
Edit	Name	txtLastName
	Text	""

续表

组件类型	属性	设置
Label	TabIndex	5
	Name	lblAddress1
	Text	Address 1:
Edit	TabIndex	6
	Name	txtAddress1
	Text	""
Label	TabIndex	7
	Name	lblAddress2
	Text	Address 2:
Edit	TabIndex	8
	Name	txtAddress2
	Text	""
Label	TabIndex	9
	Name	lblCity
	Text	City:
Edit	TabIndex	10
	Name	txtCity
	Text	""
Label	TabIndex	11
	Name	lblState
	Text	State:
ComboBox	TabIndex	12
	Name	cmbState
	Text	""
Label	Style	DropDownlist
	TabIndex	13
	Name	lblZipCode
Edit	Text	Zip Code:
	TabIndex	14
	Name	txtZipCode
	BorderStyle	""
	TabIndex	15

图 12.4 显示了添加完控件后的组合控件在设计时的外观。



图 12.4 设计时的 AddressProvider

关于控件布局要注意的一点是：对于 `cmbState` 控件的 `items` 属性，您需要添加 50 个州和 Washington, D.C. 中的一个条目。正如您所看到的，设计组合控件的控件布局与设计窗体的控件布局方法是相同的。

如果想重复使用 `cmbState` `ComboBox` 控件，那么可以在 `Forms Designer` 中右键单击控件，然后选择 `Copy`。在 `Toolbox` 中右键单击，然后选择 `Paste`。这时，具有相同州名的控件将被复制到可以在其他应用程序中重复使用的 `Toolbox` 中。

默认情况下，粘贴到 `Toolbox` 中的控件名为 `Code Fragment`。要重新命名它，使它更明确一些，您可以在粘贴的控件上右键单击，选择 `Rename Item`，然后输入一个更具描述性的控件名称。相同的功能可以用于要重复使用的代码段中。

12.3.3 创建控件属性

我们已经定义了控件的布局，下面需要为 `UserControl` 中的每个 `Edit` 及 `ComboBox` 控件添加属性，以便用户可以通过 `Properties` 窗口访问每个控件。我们还需要创建一个属性，允许用户定义显示在 `cmbState` `ComboBox` 控件中的默认州名。使用表 12.6 定义控件的属性。要确保对于除 `defaultState` 以外的其他所有属性，取消选定 `Add WFC Property` 对话框的 `Read-Only Property` 字段和 `Declare Member Variable` 字段。还要保证定义了 `Data Category` 的属性，并将它们的 `Data Type` 字段设置为 `java.lang.String`。

表 12.6 AddressProvider 属性定义

控件	添加的 WFC 属性字段	设置
cmbTitle	Name	Title
cmbTitle	Description	The current person's state
txtFirstName	Name	FirstName
txtFirstName	Description	The current person's first name
txtLastName	Name	LastName
txtLastName	Description	The current person's last name
txtAddress1	Name	Address1
txtAddress1	Description	The current person's first address line
txtAddress2	Name	Address2
txtAddress2	Description	The current person's second address line
txtCity	Name	City
txtCity	Description	The current person's city
cmbState	Name	State
cmbState	Description	The current person's state
txtZipCode	Name	ZipCode
txtZipCode	Description	The current person's Zip Code
N/A	Name	DefaultState
N/A	Description	The default state to display in the State control when initialized
N/A	Declare Member Variable	Checked
N/A	Member Variable Name	m_DefaultState

因为所有的属性都是使用相同的数据类型定义的,所以创建它们是很容易的。唯一与其他不同的属性是定义成员变量的 defaultState 属性。原因是它与可以存储属性状态的控件没有联系。

定义完属性,我们需要为它们提供代码,来获得并设置 UserControl 中控件的当前值。将下面的代码添加到 WFC Component Builder 创建的 get 和 set 方法中:



```

/* *
 * The current person's first address line
 */
public String getAddress1()
{
    return txtAddress1.getText();
}
/* *

```

```
    * The current person's second address line
    */
    public String getAddress2()
    {
        return txtAddress2.getText();
    }
    /**
    * The current person's city
    */
    public String getCity()
    {
        return txtCity.getText();
    }
    /**
    * The default state to display in the cmbState control
    * when it's initialized
    */
    public String getDefaultState()
    {
        return m_DefaultState;
    }
    /**
    * The current person's first name
    */
    public String getFirstName()
    {
        return txtFirstName.getText();
    }
    /**
    * The current person's last name
    */
    public String getLastName()
    {
        return txtLastName.getText();
    }
    /**
    * The current person's state
    */
    public String getState()
    {
        return (String)cmbState.getSelectedItem();
    }
    /**
    * The current person's state
    */
```



```
public String getTitle()
{
    return (String)cmbTitle.getSelectedItem();
}
/**
 * The current person's Zip Code
 */
public String getZipCode()
{
    return txtZipCode.getText();
}
/**
 * The current person's first address line
 */
public void setAddress1(String value)
{
    txtAddress1.setText(value);
}
/**
 * The current person's second address line
 */
public void setAddress2(String value)
{
    txtAddress2.setText(value);
}
/**
 * The current person's city
 */
public void setCity(String value)
{
    txtCity.setText(value);
}
/**
 * The default state to display in the cmbState control
 * when it's initialized
 */
public void setDefaultState(String value)
{
    m_DefaultState = value;
    setState(value);
}
/**
 * The current person's first name
 */
public void setFirstName(String value)
```

```

|
|    txtFirstName.setText(value);
|
|
|  /**
|   * The current person's last name
|   */
|  public void setLastName(String value)
|  |
|    txtLastName.setText(value);
|
|
|  /**
|   * The current person's state
|   */
|  public void setState(String value)
|  |
|    cmbState.setSelectedItem(value);
|
|
|  /**
|   * The current person's state
|   */
|  public void setTitle(String value)
|  |
|    cmbTitle.setSelectedItem(value);
|
|
|  /**
|   * The current person's Zip Code
|   */
|  public void setZipCode(String value)
|  |
|    txtZipCode.setText(value);
|
|

```

对于大多数情况,这些方法的代码是非常明确的。但是,setDefaultState 方法有些麻烦。setDefaultState 的代码将它的私有成员变量 m_DefaultState 设置为新的默认州名的 String 值。它还将调用 setState 来把 cmbState ComboBox 中的当前州名更改为新定义的默认州名。

为了完成控件属性的工作,我们需要更改 defaultState 属性的 PropertyInfo 定义,以使我们能够配置它来显示自定义值编辑器。当用户在 Properties 窗口中单击属性时,将显示我们为 defaultState 属性创建的值编辑器。用户将会在属性的值区看到一个 Ellipsis 按钮。当用户单击这个按钮时,将显示一个包含具有州名列表的 ComboBox 的对话框。用户选定州名并提交设置后,将更新 Properties 窗口,并且因为我们添加到 setDefaultState 中的代码, cmbState 控件将使用默认的州名来进行更

新。将 defaultState 属性的 PropertyInfo 声明的定义更改为以下的定义：



```
public static final PropertyInfo defaultState =
    new PropertyInfo(AddressProvider.class,
        "defaultState",
        String.class,
        CategoryAttribute.Data,
        new DescriptionAttribute(
            "The default state to display"
            + "in the State control when"
            + "it's initialized."),
        new ValueEditorAttribute(
            DefaultStateEditor.class)
    );
```

我们对 PropertyInfo 定义做的唯一更改是添加 ValueEditorAttribute。该属性警告编译器，在 Properties 窗口中显示属性时，该控件将有一个要使用的相关值编辑器。DefaultStateEditor.class 参数分配一个我们将在该实例稍后添加的值编辑器类。

12.3.4 创建 DefaultState 属性对话框

现在我们已经完成了 AddressProvider 类。稍后将返回来添加一个方法，来重新设置控件的内容和方法，以创建控件内容的地址标签。下面继续构建 defaultState 属性所需的代码来从 Properties 窗口显示它的对话框。将一个新的窗体添加到工程中，并将其命名为“DSEditor.java”。使用表 12.7 将控件添加到窗体中，并设置它们的属性。

表 12.7 DSEditor.java 组件及属性设置

组件类型	属性	设置
Form	BorderStyle	Fixed Dialog
	ControlBox	False
	MinimizeBox	False
	MaximizeBox	False
	AcceptButton	BtnOK
	CancelButton	BtnCancel
	Text	Default State Options
GroupBox	Name	grpMain
	Text	Options
Label	Name	lblDefaultState
	Text	Default State:

续表

组件类型	属性	设置
ComboBox	Name	cmbDefaultState
	Style	Dropdownlist
	Items	< < Fill with states > >
	Text	""
Button	Name	btnOK
	Text	OK
	DialogResult	OK
Button	Name	btnCancel
	Text	DialogResult
	Cancel	Cancel

图 12.5 显示了该对话框在设计时的外观。

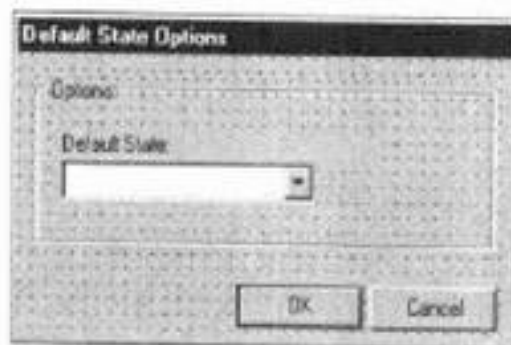


图 12.5 设计时的 DSEditor.java

AddressProvider 控件具有 cmbDefaultState ComboBox 后,需要为 cmbDefaultState ComboBox 的 items 属性指定 50 个州名。您可以从我们的控件中复制该 cmbState ComboBox 控件来简化这个过程。我们只需添加一些方法来获得 cmbDefaultState 当前选定的州名。当 DefaultStateEditor 值编辑器类初始化 DSEditor 对话框时,它将把对话框当前选定的州设置为 AddressProvider 当前的 defaultState 设置。相反,值编辑器类需要从 DSEditor 对话框获得当前选定的州来设置 AddressProvider 的 defaultState 属性。添加下面的代码,将这些方法提供给 DSEditor 窗体:



```

public String getDefaultState()
{
    return (String) cmbDefaultState.getSelectedItem();
}
public void setDefaultState(String item)
{

```

```
cmbDefaultState.setSelectedItem(item);
```

该代码非常直观,因为它或者设置具有作为参数传递的值的 `cmbDefaultState` 控件(这里是 `setDefaultState` 方法),或者返回选定的州。

12.3.5 将 DefaultStateEditor 值编辑器添加到工程中

为了完成显示 `defaultState` 属性自定义值编辑器的代码,我们需要将 `DefaultStateEditor` 添加到工程中。它是一个我们在 `defaultState` 属性的 `PropertyInfo` 定义中为 `ValueEditorAttribute` 定义的类。`DefaultStateEditor` 类是 `com.ms.wfc.core.ValueEditor` 类的一个子类。`ValueEditor` 类定义值编辑器的基本执行代码,并实现 `com.ms.wfc.core.IValueEditor` 界面。要将该类添加到工程中,我们需要添加一个新的类文件。将它命名为“`DefaultStateEditor.java`”。用下面的类定义替换该源文件的内容:



```
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.util.*;
public class DefaultStateEditor extends ValueEditor
{
    /** * Display the APDesign design page
     * */
    public void editValue(IValueAccess valueAccess)
    {
        DSEditor dialog = new DSEditor();

        /** * Set the current value of the design page to the current
         * property value
         * */
        dialog.setDefaultState(((String)valueAccess.getValue());
        // Display the design page
        int result = dialog.ShowDialog(null);
        // If the user clicked OK...set the property
        if (result == DialogResult.OK)
            valueAccess.setValue(dialog.getDefaultState());
    }

    /** * Augment the style of the value editor to include EDITVALUE */
    public int getStyle()
    {
        int style = super.getStyle();
```

```

        style |= IValueEditor.STYLE_EDITVALUE
            | IValueEditor.STYLE_NONEDITABLETEXT;
        return style;
    }

    /** * Needed to set the property in the Properties window after
     * being set by the dialog
     */
    public Object getValueFromText(String text)
    {
        return Value.toObject(text);
    }
}

```

DefaultStateEditor 的代码定义了 3 个被忽略的方法：请求更改属性时 WFC 调用的 editValue 方法，允许您在 Properties 窗口定义属性类型的 getStyle 方法，以及 Properties 窗口用来将属性的文本值转换为相当的 Object 值的 getValueFromText 方法。



注意 对于控件中的所有值编辑器，都需要忽略 getValueFromText 方法。Properties 窗口需要用这个方法从相关的值编辑器适当地设置和获得属性。

下面我们分别来看一看这些方法的代码。editValue 方法的代码从创建 DSEditor 窗体的实例开始。具有窗体实例后，代码将调用我们添加的 setDefaultState 方法使用属性的当前值来设置对话框的 ComboBox 值。传递给 editValue 的 IValueAccess 参数提供了获得和设置当前属性值的方法。调用 valueAccess 参数中 getValue 将检索 Properties 窗口中 defaultState 属性的当前值。

初始化该对话框后，它将会显示出来。返回对话框后，将比较对话框的值来查看用户是否单击了 OK 按钮。如果是，那么代码将通过使用 valueAccess 对象具有的 DSEditor getDefaultState 方法返回值的 setValue 方法来设置属性的值。

getStyle 方法的代码从获得父类的类型设置开始，这里是 ValueEditor 类。然后代码将把这些类型设置与 IValueEditor 界面的 STYLE_EDITVALUE 和 STYLE_NONEDITABLETEXT 字段进行 OR 运算。这些类型将通知 Properties 窗口，属性有一个显示文本的 Ellipsis 按钮，但是该文本不能直接从 Properties 窗口进行编辑。

getValueFromText 方法的代码比较简单：它只返回被传递的文本的 Object 版本。传递给方法的文本是 defaultState 属性的 String 值。代码使用 Value.toObject 方法把默认的州名转换为 Object 实例，然后返回它。

现在具有这个代码后，我们就完成了 defaultState 属性对话框的显示过程。下面将最后的代码添加到 AddressProvider 类中。

12.3.6 将公共方法添加到 AddressProvider 类中

为了完成我们的组合控件,我们需要将两个公共方法添加到 AddressProvider 类中。第一个方法 `resetData` 将清除组合控件中控件的内容。对于 `cmbState` 控件,如果 `defaultState` 属性具有值,那么 `cmbState` 就被重新设置为这个值。第二个方法 `getAddressLabel` 将组合控件中的所有字段组合成一个多行的、可以用来为控件中当前地址信息创建地址标签的 `String` 值。将下面的方法定义添加到 AddressProvider 类中:



```
public String getAddressLabel()
{
    String tempString = new String();
    // Write first line to string
    tempString = (String)cmbTitle.getSelectedItemAt() + " "
        + txtFirstName.getText() + " "
        + txtLastName.getText() + "\r\n";
    // Get address1 line
    tempString = tempString + txtAddress1.getText() + "\r\n";
    // If the address2 is filled with data...add it
    if (txtAddress2.getText().equals("") == false)
        tempString = tempString + txtAddress2.getText() + "\r\n";
    // Add city, state, zip line
    tempString = tempString + txtCity.getText() + ", "
        + (String)cmbState.getSelectedItemAt() + " "
        + txtZipCode.getText() + "\r\n";
    return tempString;
}

/* * Reset the controls in the control for reuseage
 */
public void resetData()
{
    cmbTitle.setSelectedIndex(-1);
    txtFirstName.setText("");
    txtLastName.setText("");
    txtAddress1.setText("");
    txtAddress2.setText("");
    txtCity.setText("");
    // Set the default state display of the State combobox
    if (m_DefaultState != null)
        cmbState.setSelectedItem(m_DefaultState);
    else
        cmbState.setSelectedIndex(-1);
    txtZipCode.setText("");
}
```


我们可以通过这最后一个操作来改进这个控件。如果您希望它使用自己的图标而不是 Visual J++ 提供的默认图标来显示在 Toolbox 中,那么可以创建一个代表您要使用的图标的 Bitmap 对象。确保位图不大于 16×16 。将位图文件命名为与控件类名相同的名称,但是要带上 .bmp 扩展名。例如,对于 AddressProvider 控件,创建一个位图,然后将它命名为“AddressProider.bmp”。将这个文件放到工程目录中。如果文件没有自动添加到工程中,可以在 Project Explorer 中单击 Show All Files 按钮,然后添加它。具有这个工程视图后,右键单击该位图文件,然后单击 Add To Project,位图将被添加到工程中。构建控件之后,位图将与它联系起来。如果您将控件打包以便其他编程工具能够使用,那么需要在相同的位置提供这个位图作为类文件或 COM DLL。

现在我们已经完成控件。构建工程,然后将它添加到 Toolbox 中。如果为控件创建了位图,那么它应该在您将控件添加到 Toolbox 中时显示出来。

12.3.7 添加测试控件的窗体

现在我们已经创建了控件,下面把它添加到窗体中来测试它。将一个新的窗体添加到工程中,并将其命名为“AddressProviderTest.java”。在 Forms Designer 中显示窗体,然后添加两个 Button 控件。将第一个 Button 控件的 text 属性设置为 Display Address,然后将它命名为“btnDisplayAddress”。为 btnDisplayAddress 的 click 事件创建一个处理程序,并将其命名为“DisplayAddress”。对于第二个 Button 控件,将它的 text 属性设置为 ResetData,并将它命名为“btnReset”。为 btnReset 创建一个 click 事件处理程序,并将它命名为“ResetData”。将 AddressProvider 控件添加到窗体中。在窗体中布置这些控件,使它看上去如图 12.6 所示。



图 12.6 开发期间的 AddressProviderTest

将下面的代码添加到 ResetData 和 DisplayAddress 方法中：



```
private void DisplayAddress(Object source, Event e)
|
|   MessageBox.show(addressProvider1.getAddressLabel());
|
|
private void ResetData(Object source, Event e)
|
|   addressProvider1.resetData();
|
```

在 Properties 窗口中单击 defaultState 属性,然后单击 Ellipsis 按钮来显示值编辑器。这时将显示 DSEditor 对话框。将 defaultState 属性更改为您所在的州,然后单击 OK 按钮。Properties 窗口将发生变化来反映默认的州,并且现在 cmbState 显示新的默认州名。

重新编译工程并运行它。显示窗体后,将信息添加到控件的字段中。单击 Display Address 按钮。这时将出现一个消息框,显示为其输入信息的人员的地址标签。单击消息框中的 OK,然后单击 Reset Data 按钮。组合控件中除了 cmbState 控件之外的其他控件内容都将被清除,而 cmbState 控件将显示当前默认的州名。

我们在该实例中创建的组合控件非常容易开发。如果去掉 defaultState 值编辑器代码,那么该控件将会给开发工作带来一些麻烦。如果可以重复使用这种信息,那么花费开发控件的这些时间将最终为您节约大量时间。虽然控件已经很好了,但是以下这些建议可以使它的功能更加强大。

- ◆ 使用我们在前面的实例中创建的 NumText 控件来限制 Zip Code 的条目数。
- ◆ 提供一些允许控件用户禁用 Title 和 Address 2 字段的属性,因为用户或许不需要使用它们。
- ◆ 像处理 defaultState 属性一样,提供一种设置默认标题的方法。

12.4 WFC 到 ActiveX 的转换

WFC 组件模型的一个最酷的功能是它的控件可以被转换为 ActiveX 控件。这使您能够把在 WFC 应用程序中使用的 WFC 控件,展示给其他语言(例如 Visual Basic 和 Visual C++ 等)或应用程序(例如 Microsoft Office 或 Microsoft Excel 等)时,进行这种转换,而不需做任何其他工作。ActiveX 在我们所知的“组件对象模型”(COM)技术中构建。COM 提供了以一种独立于语言的形式连接对象的方法。要了解 COM 的详细信息,请参见第 13 章“COM 组件开发”的有关内容。

在 Visual J++ 中,公共但不是抽象的类都可以被转换为 COM 对象,其中包

括 WFC 控件。只要有 WFC 控件的源代码,就可以通过将 COM 类的注册信息(注释标记的窗体)放到类定义的前面,来将它转换为 ActiveX 控件。

具有这种功能,WFC 和 ActiveX 控件就可以相互通用。可以创建能够转换为 ActiveX 控件的 WFC 控件,也可以将 ActiveX 控件导入到 WFC 应用程序中。这是组件模型的强大优势,并且当您要创建提高代码重用性的跨语言方案时,它将非常有用。

在该实例中,我们将把 WFC 控件导出为 ActiveX 控件。虽然我们可以将 ActiveX 控件导入到 Visual J++ 中,但是对基于 WFC 的控件进行这种操作似乎是很无聊的。您也许从来不会创建一个 WFC 组件,导出它,然后在 WFC 应用程序中使用该控件的 ActiveX 版。您可能会在 HTML 页中使用它,但是要指出的是,我们将在更为常见的情况下使用它。此外,您可以在 Visual J++ 中将 WFC 控件添加到 HTML 页中。代替使用 Visual J++ 或 HTML,导出一个 WFC 控件,然后在 Visual Basic 中使用它是很有趣的,因为它是大量 ActiveX 控件中的一个。

如果没有 Visual Basic,那么也没有问题。从 WFC 控件创建 ActiveX 控件的过程最重要的是,您将控件导入到哪一个工具,或者控件是什么,都不会有差别,它可以很容易地被导出为 ActiveX 控件,然后被客户以类似的方式使用。控件如何被导入到应用程序或开发环境中取决于程序如何执行 ActiveX 控件的导入过程。

在该实例中,我们将使用在前面实例中创建的 AddressProvider 控件,然后将它添加到 Visual Basic 窗体中。通过这个窗体,我们将访问控件的属性和方法,来说明 WFC 控件在 ActiveX 客户中的工作方式。因为这个过程对于所有的 WFC 控件是类似的,所以在使用该实例后,您将会很容易地把 WFC 控件导出为 ActiveX 控件。

12.4.1 打开控件工程

首先,我们需要在 Visual J++ 中打开 AddressProvider 控件。如果您想保留该控件原来的版本,那么可以将文件从工程复制到另一个目录中,然后从那里打开工程。打开工程后,将显示 AddressProvider.java 源文件的源代码。

12.4.2 将控件类注册为 COM 类

对于我们要展示为 ActiveX 控件的控件,需要将类声明为 COM 类。要做到这一点,将注释标记放到类声明中。该注释标记包含将类注册为 COM 对象所需的所有注册信息。当编译器看到这个标记时,它将从注释标记中获取信息并注册控件。

要定义 COM 语句标记,可以使用提供注释 COM 注册标记的 Control 模板来构建控件,也可以使用 Visual J++ 的工具来添加所需的注释标记。在 Project Properties 对话框的 COM Classes 选项卡中,只需选定要声明为 COM 对象的类名称,然后单击 OK 按钮。Visual J++ 将为您插入注释标记。对于我们的实例,只需删除 Control 模板提供的注释标记前的注释。

12.4.3 将控件打包到 COM DLL 中

为了使 ActiveX 客户能够访问您的 WFC 控件,您不仅必须将控件注册为 COM 对象,而且还要将它打包到一个可分发的文件中。要做到这一点,需要将控件的类文件、资源文件,以及其他重要文件打包到一个 COM DLL 中。进入 Project Properties 对话框,单击 Output Format 选项卡,选中 Enable Packaging 复选框,启用对话框中的控件。

要指定创建的 COM DLL,可以从 Packaging Type 下拉列表中选择 COM DLL,然后输入“AddressProvider.dll”作为 COM DLL 的名称。如果需要,可以使用 .ocx 来代替 .dll 作为 COM DLL 的扩展名,以便更明确地表示这是一个 ActiveX 控件。其余选项保持默认设置,单击 Project Properties 对话框中的 OK。现在您已经为控件定义完打包选项,可以编译工程了。Visual J++ 将编译控件的类,在“注册表”中注册类,并将控件的文件打包到一个 COM DLL 文件中。

12.4.4 注册 COM DLL

对于要在系统“注册表”中被注册为 ActiveX 控件的 COM DLL,您需要使用 Regsvr32.exe 应用程序。要运行 Regsvr32.exe,可以从【开始】菜单中选择【运行】。Regsvr32 程序以下面的格式运行:Regsvr32.exe path-to-dll,这里 path-to-dll 是 COM DLL 要注册的路径和文件名。

执行 RegSvr32 命令后,将会出现一个消息框,通知您控件注册成功或失败。如果出现错误,应该检查一下以确保使用的 COM DLL 有效,并且文件的路径正确。可以使用【运行】命令的拖动功能来使这项工作更加容易。

在系统中构建、打包并注册控件。现在可以将这个控件添加到 ActiveX 环境中。

12.4.5 创建包含控件的 Visual Basic 工程

为了测试我们导出为 ActiveX 控件的 WFC 控件的工作情况,我们需要创建

一个 Visual Basic 工程,并在 Visual Basic 中将控件添加到窗体中。另外,如果没有 Visual Basic 5.0 或更高版本,那么查看一下您是否有其他支持 ActiveX 的工具,例如 Excel 等。如果您没有使用 Visual Basic,可以参考应用程序联机手册的指导,来了解关于导入 ActiveX 控件的详细信息。

运行 Visual Basic 并创建一个 Standard EXE 工程。设置完新工程后,右键单击 Visual Basic 的 Toolbox,然后单击 Components。这时将显示 Components 对话框,它类似于 Visual J++ 的 Customize Toolbox 对话框。在列表中找到 AddressProvider 控件,选定它,然后单击 OK 按钮。控件将被添加到 Visual Basic 的 Toolbox 中。将控件添加到窗体中,并将其命名为“apMain”。

几秒钟后,控件将显示在窗体中。大多数情况下,您可以得到与 Visual J++ 中相同的功能。在窗体中,添加两个按钮,并将它们命名为“btnReset”和“btnDisplayLabel”。将 btnReset 控件的 Caption 属性设置为 Reset Data。同样地,对于 btnDisplayLabel,将它的 Caption 属性设置为 Display Data。为两个控件创建 click 事件处理程序。图 12.7 显示了 Visual Basic 中添加完所有控件的窗体在开发时的外观。



图 12.7 Visual Basic 窗体中的 AddressProvider 控件

将下面的 Visual Basic 代码添加到方法中:

```
Private Sub btnDisplayLabel_Click()  
    MsgBox "The address is " + vbCrLf + apMain.getAddressLabel,...  
    vbInformation
```




```
End Sub  
Private Sub btnReset_Click()  
    apMain.resetData  
End Sub
```

该代码与我们在文本窗体的 Java 版中编写的代码基本相同。唯一要做的工作是在 Properties 窗口中将 defaultState 属性设置为 Washington。将 WFC 创建的 ActiveX 控件导入到 Visual Basic(或其他 ActiveX 客户)中时有一个限制,您添加到控件中的任何值编辑器在 ActiveX 版中都不能使用。虽然 AddressProvider 控件的 WFC 版不允许您将 defaultState 的值键入到 Properties 窗口中,但是您可以在 ActiveX 控件版中做到这一点。

在 Visual Basic 中运行该程序,并执行与我们在前面实例中进行的相同类型的测试。如果您不知道它是 Visual Basic,那么您将很难辨别显示控件及与其交互的语言。如果您正在设计 WFC 控件,并且计划将它们导出为 ActiveX 控件,那么您应该设计它们,以便它们可以在两种环境中以相同的方式操作。这也许意味着在 WFC 控件版本中可以使用较少的值编辑器,或者提供多种访问所需属性的可选方法。

因为该实例不要求编译工程,所以这里没涉及太多的附加功能。唯一的建议是,试着将我们在该实例中创建的其他控件导出到 Visual Basic 或其他 ActiveX 客户,以查看处理 WFC 控件的最佳选择。



更上一层楼

在本章中,我们介绍了 WFC 组件模型,以及创建可以添加到 WFC 应用程序中的自定义控件所涉及的工作。我们从创建作为现有 WFC 控件子类的控件,以及添加一些附加功能、属性和事件的控件开始。在创建该控件时,学到如何结合使用 WFC Component Builder 和 Class Outline 来简化属性、事件及方法忽略的创建过程。第二个实例说明了如何创建处理它们自己的绘制例程的控件。虽然我们的 Clock 实例很简单,但是它确实对理解如何在控件中执行双缓冲绘制,以及如何从头创建控件有很好的帮助。我们创建的第三个也是最后一个控件演示了如何使用 UserControl 类及其他 WFC 控件来创建组合控件。该实例还教我们学会了如何为控件中的属性创建自定义值编辑器。

最后,为了演示如何将现有的 WFC 控件导出为 ActiveX 控件,然后使用它,

我们导出了 AddressProvider 控件,并使用少量代码将它添加到 Visual Basic 窗体中。最后启动 Visual Basic 工程,像我们在 WFC 窗体中所做的那样,与我们添加到原来的 AddressProvider 实例中的控件进行交互,以测试该控件。

要了解关于 WFC 组件模型高级功能及常规 WFC 控件开发的详细信息,请参考 Visual J++ 联机文档“Writing WFC Controls”的有关内容。

第 13 章 COM 组件开发

知识要点:

- ◆ COM、DCOM 的基本概念
- ◆ 如何创建 COM 服务器
- ◆ 如何使用 COM 服务器
- ◆ 与数据库相关的 COM 组件
- ◆ 如何使用第三方 COM 组件
- ◆ 如何使用 Microsoft Word COM 组件

在本章中,我们将探讨 COM 在应用程序开发过程中的不同作用。我们将从创建一个提供棒球队统计功能的 COM 组件开始。为了说明通过 COM 展示常见的用户界面,将创建一个展示启动屏幕和 Find And Replace 对话框的 COM 组件。您将会看到,当您编写了具有可重复使用的代码对象时,可以通过 COM 向其他应用程序展示。

数据库访问是开发应用程序的一个重要方面。为了说明 COM 与数据库的交互方式,我们将创建一个具有在数据库中搜索客户的用户界面的 COM 组件。用户选定特定的客户后,COM 组件将返回客户应用程序可以使用的 DataSource 对象。

为了说明程序如何在其他应用程序中使用 COM 组件来执行特定的任务,我们将创建一个使用 Microsoft Word 拼写检查功能进行 WFC 应用程序拼写检查的程序。

学习完本章中的各种实例后,相信您将可以在程序开发中轻松地使用 COM。



光盘 参阅本书配套光盘中的【COM 组件】,可交互学习与本章相关的知识。

13.1 关于 COM 的相关概念

代码重用的困境

面向对象编程语言的出现,比如 C++ 和 Java,使开发人员能够创建可以在多个应用程序中重复使用的对象,而不仅仅是独立的程序,这与面向过程的编程语言的重用方式有本质的区别,面向过程的编程语言几乎没有支持代码重用的语言结构,并且缺乏统一的接口。而在面向对象编程语言中,直接提供了支持代码重用的语言结构,即对象。提取和继承是创建可重用对象的面向对象程序设计的关键部分:提取使您能够开发可以独立使用的对象;继承允许您接收来自对象的功能,用来开发新的对象。通过结合使用这两种功能,不仅可以开发独立的对象,也可以更改对象以适应特殊的需要。因为让一个对象完成所有的工作是不可能的,所以这种适应性非常重要。

为什么使用 COM

我们可以将一些对象组合成组件来创建应用程序,开发对于特定类型的数据来执行特定任务的组件,组合其他组件和少量代码来将它们“粘合”在一起,使整个应用程序都是由组件组成,而没有其他内容。可以将这些组件看成七巧板的部件,每一块在组合完整的拼图时都有它特定的用途。

希望使用基于组件模型的开发人员面对的问题是如何让“粘合”类型的兼容性更好。理想情况下,应该使用可以被所有面向对象的编程语言使用的编程语言来创建组件。因为很多公司使用不同的编程语言编写应用程序,所以只有在您可以将组件导入到其他语言中时,组件级的开发的优越性才得以体现。

为了适应在各种应用程序和编程环境下使用组件的需要,Microsoft 创建了一种“组件对象模型”(Component Object Model,COM)技术。COM 是一种独立的编程模型,它允许您开发能够用来创建应用程序的组件。COM 定义了组件与其他组件通信和交互的方式。因为 COM 是一种组件模型,所以可以在其基础上开发新的技术。事实上,诸如 ActiveX 及对象链接和嵌入(OLE)等技术都可以构建在 COM 上。

通过 COM 创建的组件展示我们所说的 COM 接口。COM 接口通过组件展示的公共方法来定义。通过 COM 接口展示的方法使其他对象能够操纵 COM 组件。只要组成组件 COM 接口的方法标记保持完整,您就可以从应用程序中删除 COM 组件,或者修改它,以及在不重新编译应用程序的情况下将它重新合并到

应用程序中。这类似于展示“造型”的七巧板部件。如果这个部件(或组件)保持不变,那么它可以重新与另外的拼板部件组合来重新组成原来的图形。

这种 COM 功能在组件开发和代码重用性方面是非常重要的:我们可以创建 COM 组件,并为它添加功能。只要 COM 组件的接口保持完整,就可以使用这种添加到原来应用程序中的功能。因为 COM 是一个独立的实体,所以在 COM 组件中所做的更改也会在使用它的所有其他应用程序中得到反映。创建由 COM 组件组成的应用程序将更易于调试和维护,也更易于为其添加功能。

DCOM

除了创建可以在多个应用程序中使用的组件之外,还可以使用 COM 来创建以后可以发布和可从其他系统中的应用程序访问的组件。这种技术就是我们所说的“分布式组件对象模型”(Distributed Component Object Model, DCOM)。DCOM 是一个系统中的 COM 对象与其他系统中任意数目的客户应用程序之间的通信协议。通常情况下,DCOM 作为客户应用程序和其他工具(如 SQL 服务器等)之间的通道使用。除了作为两个应用程序的中间层之外,DCOM 还提供了简单的维护方法。通过将 COM 放到服务器中,您可以在一个地方修改组件,然后在每个系统的每一个客户应用程序中得到反映,而不用重新编译使用该组件的应用程序。

13.2 COM 服务器

我们经常需要创建对不同的应用程序提供不同功能的组件。例如,创建一套电子应用程序,并且希望对于每个应用程序提供一组相同的电子计算,而不用手工将它编写到每个应用程序中,那么您可以创建一个展示方法的 COM 服务器来执行这些计算。而后如果在这些计算中发现了小毛病,那么可以修改方法的执行代码,而不用更改每一个使用该计算的程序。

在本节的例子中,我们将创建一个展示棒球统计数据的 COM 服务器。COM 服务器将展示棒球运动员平均击中率、触垒百分比及本垒打百分比统计的计算方法。COM 服务器还将展示每个投手平均投杀能力(ERA)的计算方法。在接下来的例子中,将创建一个说明如何使用棒球统计 COM 对象的客户应用程序。

13.2.1 使用 COM 模板创建工程

Visual J++ 提供了两种创建 COM 组件的方法。第一种是通过 COM DLL 工

程模板。它创建一个具有事先已经配置为 COM 对象的类的工程。该模板还将配置工程打包到 COM DLL 中。(关于 COM DLL 的详细内容,将在稍后的例子中介绍。)为了使 Visual J++ 能够通过 COM 将 Java 类展示为组件,类需要在它的声明前有一个 @com.register 注释标记。在编译过程中,Visual J++ 查找这个注释标记,并使用它在“注册表”中将类注册为 COM 对象。

创建 COM 组件的第二种方法是使用工程 Properties 对话框中的 COM Classes 选项卡。在该对话框中,可以选择在工程中要通过 COM 展示的任何公共的、非抽象的类。Visual J++ 将 COM DLL 工程模板添加的相同类型的注释标记添加到在对话框中指定的每一个类中。

下面我们使用 COM DLL 工程模板来创建棒球统计的 COM 组件。可以在 New Project 对话框的 Components 节点中发现具有其他 Visual J++ 工程模板的模板。使用该模板创建一个新的工程,并将它命名为“BaseballMath”。默认情况下,该工程将包含一个名为“Class1.java”的类。将它重新命名为“BaseballMath.java”,同时在源文件中将带有“Class1”的实例重新命名为“BaseballMath”。

13.2.2 在类中添加方法

现在我们已经创建了工程,并准备了 COM 类,下面需要添加将要展示和组成 COM 接口的方法。将下面的代码添加到 BaseballMath 的类定义中:



```

/* * Calculates a batter's batting average
 */
public float getBattingAverage(int atBats, int totalHits)
{
    // Ensure we have no 0 values to divide
    if (totalHits == 0 || atBats == 0)
        return 0.0f;
    else
    {
        float tempHits = totalHits;
        float tempAtBats = atBats;

        return tempHits/tempAtBats;
    }
}

/* * Calculates a batter's slugging percentage
 */
public float getSluggingPercentage(int singles, int doubles,
                                    int triples, int homers,
                                    int atBats)

```

```

    float result;
    float returnVal;
    float tempAtBats = atBats;

    // Add singles to total
    result = singles;
    /* Multiply number of doubles by 2 to indicate
     * number of bases. Add to result. */
    result = result + (doubles * 2);
    /* Multiply number of triples by 3 to indicate
     * number of bases. Add to result. */
    result = result + (triples * 3);
    /* Multiply number of homers by 4 to indicate
     * number of bases. Add to result. */
    result = result + (homers * 4);
    // Ensure we have no 0 values to divide
    if (result == 0 || atBats == 0)
        return 0.0f;
    else
        /* Calculate result based on total bases divided
         * by the number of at bats. */
        returnVal = result / tempAtBats;

    return returnVal;
}

/* * Calculates a batter's on-base percentage
 * */
public float getOnBasePercentage(int hits, int walks,
                                int hitByPitch,
                                int sacrificeFlies,
                                int atBats)
{
    // Add hits, walks, and hit-by-pitch counts for player
    float positive = hits + walks + hitByPitch;
    /* Add at bats, walks, hit-by-pitches,
     * and sacrifice flies for player. */
    float negative = atBats + walks
                    + hitByPitch + sacrificeFlies;

    if (negative == 0 || positive == 0)
        return 0.0f;
    else
        return positive / negative;
}

/* * Calculates a pitcher's earned run average (ERA)
 * */

```

```

public float getEarnedRunAverage(int runs, float innings)
{
    // Ensure we have no 0 values to divide
    if (runs == 0 || innings == 0)
        return 0.0f;
    else
        return ((runs * 9) / innings);
}

```

我们添加的第一个方法 `getBattingAverage` 通过用运动员击中球的总次数除以击球的总次数来计算运动员的平均击中率。棒球统计通常不返回百分比,而是使用一个小于 1 的数。`getBattingAverage` 将返回诸如 .200 和 .304 等的值。

`getBattingAverage` 方法从确保没有值为 0 的参数开始。如果有,那么代码将返回 0 值。如果是有效值,那么代码将继续在本地浮点变量中存储 `atBats` 和 `totalHits` 参数。这会使除法运算更加方便。如果只是除以两个值,那么将得到 0,因为整数相除不能产生浮点数。因为大多数棒球统计是以整数值存储的,所以将整数作为参数接受,然后以浮点值存储它们来进行计算。存储完值之后,代码最后以击中次数除以击球次数来返回结果。这一切非常简单。

我们添加的第二个方法是 `getSluggingPercentage`。它将生成一个类似于平均击中率的值,表示击球手返回本垒的百分比。例如,击出很多本垒打的击球手可能有很高的触垒百分比,因为它们每打出一个本垒打就将触及 4 个垒。`getSluggingPercentage` 方法将运动员触及一垒、二垒、三垒和本垒的次数,以及击中次数作为参数接受。

该方法的代码定义一些在计算中使用的浮点变量,以及将 `atBats` 参数存储为一个名为“tempAtBats”的浮点数,然后将 `singles` 参数的值添加到 `result` 变量中。`result` 变量存储运动员触垒的总数。代码用 2 乘以 `doubles` 参数,用 3 乘以 `triples` 参数,用 4 乘以 `homers` 参数,并将每个结果累加到当前的 `results` 值中。代码得到触垒总数后,它将检查确保 `atBats` 和 `results` 变量不等于 0。代码最后用 `tempAtBats` 除以 `result`,并返回结果。

`getOnBasePercentage` 方法计算运动员在每次击中后触垒次数的百分比。代码将接受一些参数,其中包括运动员的击球数、出局数、每组投球的击中数、安打数,以及击中数等。代码把 `hits`、`walks`、`hitByPitch` 参数存储到一个名为 `positive` 的浮点变量中。代码还将 `atBats`、`walks`、`hitByPitch` 及 `sacrificeFlies` 参数添加到第二个名为 `negative` 的浮点变量中。与在其他方法中一样,代码将检查确保它的浮点变量不等于 0。当确定存在有效值时,代码将用 `negative` 除以 `positive`,并返回结果。

我们还将添加 `getEarnedRunAverage` 方法,它将计算允许投手在一组投球中被跑垒的次数。与其他计算不同,这个方法将产生一个大于 1 的值。该方法接

受两个参数:被跑垒的总次数(投手造成的跑垒数);投手投球的总局数。

因为教练可以在一局当中换下投手,所以我们需要动态存储投球局数。该方法的代码确定 runs 参数或 innings 参数是否等于 0。如果这两个参数都等于 0,那么像其他方法一样,代码将返回 0。否则,代码将被跑垒数乘以 9,然后用投球局数除以所得的积。然后该值作为方法的返回值返回。

现在代码已经完成,下面可以编译工程了。因为我们是使用 COM DLL 模板创建的这个工程,所以在编译该工程时,类文件将被打包到一个 COM DLL 文件中。Visual J++ 将这个 DLL 的名称设置为工程的名称。在我们的实例中,编译时定义的 DLL 名为 BaseballMath.dll。对于访问 COM 组件的其他应用程序,必须包含在 Windows 的 EXE 文件或 COM DLL 文件中。如果希望将 COM 组件展示为 HTML 页面,也可以将组件打包到一个 CAB 文件(*.cab)中。

COM 组件有两种类型:

- ◆ **进程内组件。**包含在 COM DLL 或 .cab 文件中,并且在没有客户应用程序使用的情况下不能实例化。进程内组件被绑定到启动它并在客户进程终止时结束的进程中。进程内组件比进程外组件与它的客户应用程序有更紧密的关系,因为进程内组件和它的客户在相同的环境下操作,并且不需要远程调用过程来进行通信。
- ◆ **进程外组件。**包含在 EXE 文件中,并且在与客户应用程序不同的环境下操作。因此,如果客户应用程序无意中被终止,那么进程外组件可能会被保留在内存中。通常情况下,当包含在一个较大的应用程序中,以及希望通过组件展示应用程序功能时,可能需要创建进程外组件。Microsoft Excel 是进程外组件一个很好的例子,因为它直接通过它的 EXE 文件来展示它的对象模型。

在本书所有的实例中,都将使用进程内组件,因为它们更为常见。

编译时,在创建 COM DLL 之前,Visual J++ 执行两个任务。首先,它根据类中的信息和类的 @com.register 注释标记创建一个类型库。其次,Visual J++ 使用这个类型库来将类注册到“注册表”中。完成这些过程后,COM 组件可以通过其他应用程序和编程环境来访问。在接下来的实例中,我们将创建一个客户应用程序来演示 COM 组件的用法。

13.3 WFC COM 客户应用程序

Visual J++ 对在 Visual J++ 和其他兼容 COM 的应用程序中开发的 COM 组件提供了完全的支持。Visual J++ 用来访问这些组件的机制就是我们所说的

“COM 类封装程序”。回顾在第 9 章“Java 应用程序编程进阶”讨论导入 ActiveX 控件的内容, Visual J++ 创建了一些允许访问 ActiveX 控件属性和方法的类封装程序。因为 ActiveX 是在 COM 上构建的, 所以在 Visual J++ 使用相同类型的类封装程序来访问 COM 组件。导入 COM 服务器和 ActiveX 控件之间的区别是创建 COM 封装程序类的过程。使用前面实例中创建的 COM 组件, 我们将讨论如何将 COM 组件导入到标准的 WFC 应用程序中。

13.3.1 创建 WFC 应用程序工程

首先使用 Empty Project 模板创建一个新的工程, 并将其命名为“Baseball-MathClient”。将窗体添加到工程中, 并将其命名为“BaseballClient.java”。将它的 text 属性设置为 Baseball Math Client Application, 将它的 borderStyle 属性设置为 Fixed Dialog, 并将它的 maximizeBox 属性设置为 false。该窗体将包含 4 个选项卡, 我们在 BaseballMath COM 组件中创建的每一个方法都有一个选项卡。

13.3.2 设计窗体的用户界面

在 Forms Designer 中打开 BaseballClient 窗体, 然后将 TabControl 添加到窗体中, 并将该控件命名为“tabMain”。通过单击 tabs 属性值区中的 Ellipsis 按钮, 将 4 个 TabPage 添加到控件中。使用表 13.1 来设置添加的 4 个 TabPage 的属性。

表 13.1 BaseballClient.java TabPage 属性设置

TabPage 索引号	属性	设置
0	name	tpBatting
	text	Batting Average
1	name	tpSlugging
	text	Slugging Percentage
2	name	tpOnBase
	text	On-Base Percentage
3	name	tpERA
	text	ERA

添加完选项卡后, 使用表 13.2 将控件添加到每个选项卡中。这些设置决定了控件将要被放到哪一个 TabPage 中。

表 13.2 BaseballClient.java 组件及其属性设置

选项卡	组件类型	属性	设置
tpBatting	Label	name	lbBAAtBats
		text	Number of At Bats:
		tabIndex	0
tpBatting	Edit	name	txtBAAtBats
		text	" "
		tabIndex	1
tpBatting	Label	name	lbBAHits
		text	Hits:
		tabIndex	2
tpBatting	Edit	name	txtBAHits
		text	" "
		tabIndex	3
tpBatting	Label	name	lbBAResult
		text	Result:
		tabIndex	4
tpBatting	Label	name	lbBAValue
		text	" "
		borderStyle	Fixed 3d
		tabIndex	5
tpBatting	Button	name	btnBACalculate
		text	Calculate
		tabIndex	6
tpSlugging	Label	name	lbISingles
		text	Singles:
		tabIndex	0
tpSlugging	Edit	name	txtSingles
		text	" "
		tabIndex	1
tpSlugging	Label	name	lbIDoubles
		text	Doubles:
		tabIndex	2
tpSlugging	Edit	name	txtDoubles
		text	" "
		tabIndex	3

续表

选项卡	组件类型	属性	设置
tpSlugging	Label	name	lblTriples
		text	Triples:
		tabIndex	4
tpSlugging	Edit	name	txtTriples
		text	" "
		tabIndex	5
tpSlugging	Label	name	lblHomeRuns
		text	Home Runs:
		tabIndex	6
tpSlugging	Edit	name	txtHomeRuns
		text	" "
		tabIndex	7
tpSlugging	Label	name	lblSPAtBats
		text	Number of At Bats:
		tabIndex	8
tpSlugging	Edit	name	txtSPAtBats
		text	" "
		tabIndex	9
tpSlugging	Label	name	lblSPResult
		text	Result:
		tabIndex	10
tpSlugging	Label	name	lblSPValue
		text	" "
		borderStyle	Fixed 3d
		tabIndex	11
tpSlugging	Button	name	btrSPCalculate
		text	Calculate
		tabIndex	12
tpOnBase	Label	name	lblOBHits
		text	Hits:
		tabIndex	0
tpOnBase	Edit	name	txtOBHits
		text	" "
		tabIndex	1
tpOnBase	Label	name	lblWalks
		text	Walks:
		tabIndex	2

续表

选项卡	组件类型	属性	设置
tpOnBase	Edit	name	txtWalks
		text	" "
		tabIndex	3
tpOnBase	Label	name	lblHitByPitch
		text	Hit By Pitch:
		tabIndex	5
tpOnBase	Edit	name	txtHitByPitch
		text	" "
		tabIndex	4
tpOnBase	Label	name	lblSacrifices
		text	Sacrifice Flies:
		tabIndex	5
tpOnBase	Edit	name	txtSacrifices
		text	" "
		tabIndex	6
tpOnBase	Label	name	lblOBAtBats
		text	Number of At Bats:
		tabIndex	7
tpOnBase	Edit	name	txtOBAtBats
		text	" "
		tabIndex	8
tpOnBase	Label	name	lblOBResult
		text	Result:
		tabIndex	9
tpOnBase	Label	name	lblOBValue
		text	" "
		borderStyle	Fixed 3d
		tabIndex	10
tpOnBase ate	Button	name	btnOBCalcul
		text	Calculate
		tabIndex	11
tpERA	Label	name	lblEarnedRuns
		text	Earned Runs:
		tabIndex	0
tpERA	Edit	name	txtEarnedRuns
		text	" "
		tabIndex	1

续表

选项卡	组件类型	属性	设置
tpERA	Label	name	lblInnings
		text	Innings:
		tabIndex	2
tpERA	Edit	name	txtInnings
		text	""
		tabIndex	3
tpERA	Label	name	lblERResult
		text	Result:
		tabIndex	4
tpERA	Label	name	lblERValue
		text	""
		borderStyle	Fixed 3d
		tabIndex	5
tpERA	Button	name	btnERCakulate
		text	Calculate
		tabIndex	6

图 13.1 ~ 图 13.4 演示了当正确放置完控件并设置完其属性时的各个选项卡的外观。

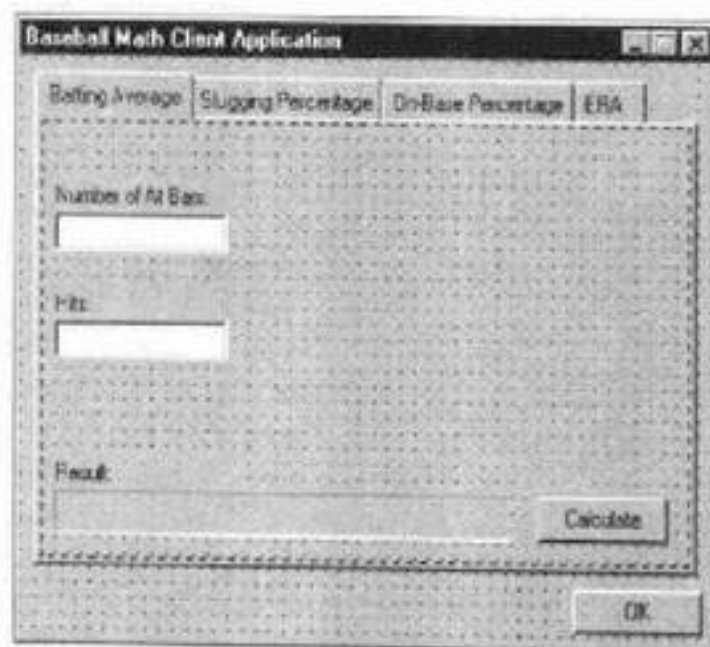


图 13.1 设计时的 Batting Average TabPage

The screenshot shows a Windows application window titled "Baseball Math Client Application". It has four tabs: "Batting Average", "Slugging Percentage", "On-Base Percentage", and "ERA". The "Slugging Percentage" tab is selected. Inside the tab, there is a dotted rectangular area containing the following elements:

- Labels and input fields: "Singles:" with an input box, "Home Runs:" with an input box, "Doubles:" with an input box, and "Number of At Bats:" with an input box.
- A label "Triples:" with an input box below it.
- A label "Result:" followed by a large text input field.
- A "Calculate" button to the right of the "Result" field.

Below the dotted area, there is an "OK" button.

图 13.2 设计时的 Slugging Percentage TabPage

The screenshot shows the same "Baseball Math Client Application" window, but with the "On-Base Percentage" tab selected. The layout is similar to the previous tab, with a dotted rectangular area containing:

- Labels and input fields: "Hits:" with an input box, "Sacrifice Flies:" with an input box, "Walks:" with an input box, and "Number of At Bats:" with an input box.
- A label "Hit-By-Pitch:" with an input box below it.
- A label "Result:" followed by a large text input field.
- A "Calculate" button to the right of the "Result" field.

Below the dotted area, there is an "OK" button.

图 13.3 设计时的 On-Base Percentage TabPage

为了完成窗体用户接口的设计,将按钮添加到 TabControl 的下面,并将其命名为 btnOK,将它的 text 属性设置为 OK。

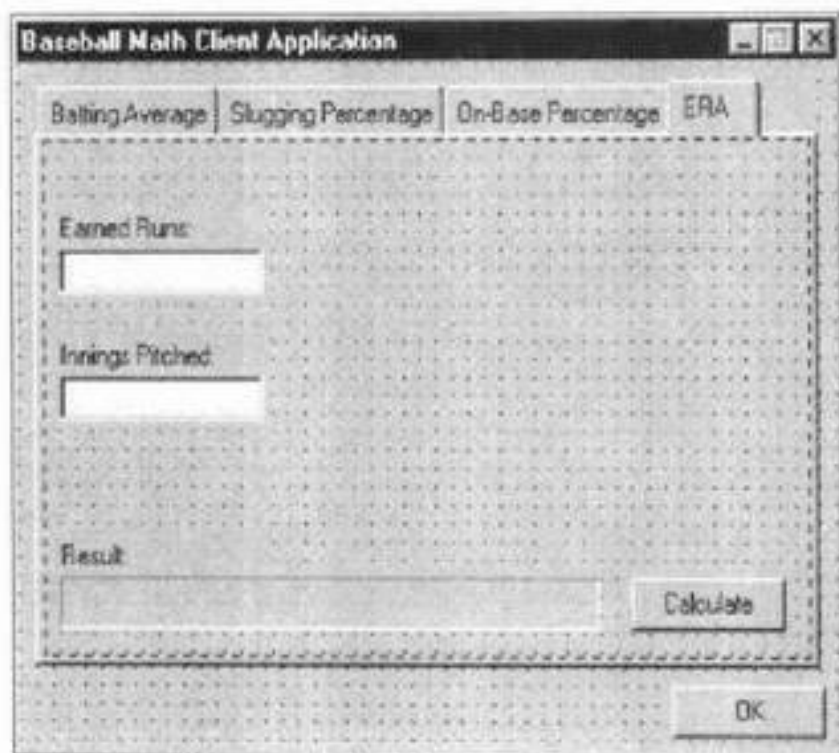


图 13.4 设计时的 ERA TabPage

13.3.3 导入 COM 组件

设计完窗体用户界面,需要将 BaseballMath COM 组件导入到应用程序中。可以从 Project 菜单中选择 Add COM Wrapper,或者在 Project Explorer 中单击右键,选择 Add,然后选择 Add COM Wrapper 菜单项来导入它。这两种方法都将显示如图 13.5 所示的 COM Wrappers 对话框。

COM Wrappers 对话框显示了系统中当前注册的 COM 组件列表。当您从列表选定 COM 组件时,对话框下面的区域将显示关于该组件的重要信息。选定 BaseballMath COM 组件,然后单击 OK 按钮。Visual J++ 将在工程中创建一个名为 baseballmath 的新数据包。在该数据包中,它将创建两个 Java 源文件:BaseballMath.java 和 BaseballMath_Dispatch.java。第一个源文件是 BaseballMath COM 组件的类封装程序,第二个表示 COM 组件的分发接口。通常情况下,使用标准的类封装程序。

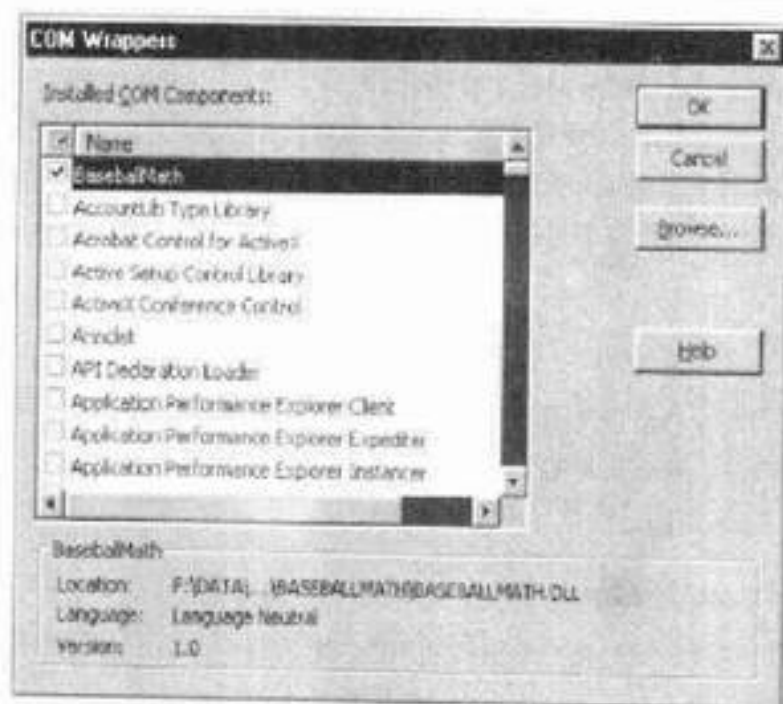


图 13.5 COM Wrappers 对话框

如果您计划通过 DCOM 使用 COM 组件,或者 COM 组件是一个进程外组件,那么您将无法创建它的实例,而只能通过分发接口的公共方法来引用组件。因为我们的应用程序将 BaseballMath COM 组件作为进程内组件使用,所以我们将使用 BaseballMath 类。好在有类封装程序,一旦将 COM 组件导入到应用程序中,就可以像应用程序中的其他组件那样使用它。



注意 除非需要调试 COM 组件类封装程序的源代码,否则不要修改它。如果您修改了类封装程序的代码,那么将有可能无法与基础的 COM 组件进行通信。

13.3.4 添加支持代码

现在我们已经将 BaseballMath COM 组件导入到应用程序中,下面我们将把支持代码添加到客户应用程序中,以查看该 COM 组件的实际效果。在创建事件处理程序和代码之前,先将两个导入语句添加到 BaseballClient.java 源文件的前面:

```
import baseballmath.*;
import com.ms.wfc.util.*;
```

导入 com.ms.wfc.util 数据包将使我们能够使用它的 Value 类功能来将值转

换成文本。导入 baseballmath 数据包将使我们能够引用 COM 封装程序类,而不将数据包名称附加到每个组件引用中。

为了引用 COM 对象,需要像其他对象那样在代码中创建它的实例。将一个 COM 对象实例定义为客户类的私有成员变量是一个很好的做法,这将使您能够在整个应用程序中只维护一个对象。将下面的代码添加到 BaseballClient 类的定义前面:

```
private BaseballMath bball = new BaseballMath();
```

现在已经定义了 COM 组件的实例,下面我们将为客户应用程序创建事件处理程序。这里只有两个事件处理程序。首先为 btnOK Button 控件的 click 事件创建一个处理程序,然后添加下面的代码:

```
Application.exit();
```

需要创建的第二个事件处理程序是为每个选项卡中的所有 Calculate 按钮创建的。因为 Calculate Button 控件执行基本相同的任务,所以将所有任务都集中到一个事件处理程序会更容易一些。创建一个名为 Calculate 的事件处理程序,然后将它附加到每个按钮的 click 事件中。将下面的代码添加到 Calculate 方法中:



```
int curTab = tabMain.getSelectedIndex();
// Determine the tab that is currently selected
switch(curTab){
case 0: // Batting Average
    int atBats = Value.toInt(txtBAAtBats.getText());
    int hits = Value.toInt(txtBAHits.getText());

    lblBAValue.setText(String.valueOf(
        bball.getBattingAverage(atBats,
                                hits)));

    break;
case 1: // Slugging Percentage
    int singles = Value.toInt(txtSingles.getText());
    int doubles = Value.toInt(txtDoubles.getText());
    int triples = Value.toInt(txtTriples.getText());
    int homers = Value.toInt(txtHomeRuns.getText());
    int SPAtBats = Value.toInt(txtSPAtBats.getText());

    lblSPValue.setText(String.valueOf(
        bball.getSluggingPercentage(singles,
                                    doubles,
                                    triples,
                                    homers,
```

```

SPAtBats)));
break;
case 2: // On-Base Percentage
    int OBhits = Value.toInt(txtOBHits.getText());
    int walks = Value.toInt(txtWalks.getText());
    int hbp = Value.toInt(txtHitByPitch.getText());
    int sacs = Value.toInt(txtSacrifices.getText());
    int OBAtBats = Value.toInt(txtOBAtBats.getText());

    lblOBValue.setText(String.valueOf(
        bball.getOnBasePercentage(OBhits,
                                    walks,
                                    hbp,
                                    sacs,
                                    OBAtBats)));

    break;
case 3: // ERA
    int er = Value.toInt(txtEarnedRuns.getText());
    float innings = Value.toFloat(txtInnings.getText());
    lblERValue.setText(String.valueOf(
        bball.getEarnedRunAverage(er, innings)));

    break;
}

```

Calculate 的代码从获得控件当前选定的选项卡的索引开始。选择语句然后判断这个值。对于每一个选项卡, Value.toInt 方法都将 Edit 控件的值从文本转换为整数。对于 ERA 选项卡的 innings 值, Value.toFloat 方法将在浮点变量中存储值。一旦创建了用于计算的值, 代码就会使用适当的方法来调用 COM 服务器的实例。然后代码将 COM 组件方法的返回值分配给指定选项卡中适当的值标签。因为返回的值都是浮点值, 所以 String 类的 valueOf 方法将把它们转换为 String。

现在 COM 客户应用程序已经完成, 下面可以编译并运行它。显示客户窗体后, 选择 Batting Average 选项卡, 然后在 Number Of At Bats 字段中输入“315”, 在 Hits 字段中输入“90”。单击 Calculate 按钮, 您将在 lblBAValue Label 控件中看到平均击中率的值为 .2857143。试着进行其他选项卡中的计算。最后单击 OK 按钮, 退出应用程序。

13.4 用户接口组件

通过 COM 展示方法是将经常使用的方法合并到一个其他应用程序和编程

环境可以访问的组件中的好途径。经常需要在多个工程中使用常见的用户接口。使用与创建非可见组件(BaseballMath.dll)相同的过程,可以创建能够显示任何类型的用户接口的 COM 组件。例如,可以使用这个功能来创建一个 COM 组件,其中包含一个公司的应用程序套件中所有经常使用的用户接口。当其中一个应用程序需要显示工程(如启动屏幕或进程对话框)时,它可能只是通过一个 COM 组件来访问它们。这就体现了 COM 接口功能的方便性。如果公司希望更改它所有应用程序产品中的公司徽标,那么它可以只更新 COM 组件中的基本代码,而不必重新编译应用程序。

在本节中,我们将研究 COM 展示用户接口的能力。在第 8 章“Java 应用程序编程入门”中,创建了一个启动屏幕和一个 Find And Replace 对话框。我们将把它设计为具有代码重用性。通过使用一个常见的用户接口 COM 组件展示它们来使两个对话框能够在设计过程中任意使用。这个 COM 组件使我们能够在任何应用程序中访问这些经常使用的对话框。

13.4.1 创建工程

首先使用 COM DLL 工程模板创建一个新的 COM DLL 工程,并将其命名为 FormReuse。将类文件名重新命名为 FormReuse.java。确保用 FormReuse 替换所有 Class1 实例。

13.4.2 将对话框添加到工程中

现在已经创建了工程,下面需要将添加在第 8 章中创建的启动屏幕和现成的 Find And Replace 对话框的源文件。如果您没有创建它们,那么现在可以添加对话框的源文件,可以使用本书附带的光盘来添加(在 Samples \Chap13 \Toolbox-Example 目录下)。因为这些组件都是独立设计的,所以不必修改它们的设计或接口。

13.4.3 将方法添加到 COM 类中

您或许会问:“为什么将窗体展示为 COM 类,而不使用独立的 COM 类?”原因是它可以在 COM 组件的内部对象和 COM 接口之间提供一个提取层。当处理从 com.ms.wfc.ui.Form 获得的类时,尤其是这样。Form 类提供了很多公共方法。请记住,Visual J++ 将把所有公共的、非抽象方法展示为 COM 组件的 COM 接口的方法。方法的数目会令组件的终端用户无法承受。下面将一个提取层添

加到 COM 组件中。请把下面的方法定义添加到 FormReuse 类定义中：



```

/* * Creates an instance of the splash screen with predefined data */
public void showSplash(String companyName,
                        String logoImage,
                        String backgroundImage,
                        String copyright,
                        String warningText,
                        String userName,
                        String productName,
                        String version,
                        int delay)
{
    SplashScreen tempSplash = new
    SplashScreen(productName,
                  version,
                  copyright,
                  userName,
                  companyName,
                  warningText);
    tempSplash.setSplashBackground(backgroundImage);
    tempSplash.setLogoImage(logoImage);
    tempSplash.showSplash(delay);
}

/* * Creates an instance of the Find And Replace window
*/
public void showFindAndReplace(com.ms.wfc.ui.Form parent,
                               com.ms.wfc.ui.Edit edit)
{
    FindAndReplace tempFind = new FindAndReplace();
    tempFind.setOwner(parent);
    tempFind.setEditControl(edit);
    tempFind.show();
}

```

因为这个 COM 组件在它运行过程中包含不确定数目的可重用对话框，所以需要限制通过 COM 展示的属性方法数。正如您从代码中看到的那样，我们只展示两个简单的访问对话框的方法：一个用来显示启动屏幕，另一个用于 Find And Replace 对话框。如果添加像 getTitle 或 setVersion 等方法，那么也许会与稍后添加到 COM 组件中的其他对话框发生冲突。

我们添加的第一个方法是 showSplash。该方法采用与启动屏幕对话框的构造函数、logoImage、backgroundImage 及 delay 参数相同的参数。该方法的代码创建启动屏幕对话框，传递组件的 showSplash 参数相关的子集，然后调用启动屏幕实

例具有 `backgroundImage` 参数的 `setSplashBackground` 方法,以及实例具有 `logoImage` 参数的 `setLogoImage` 方法。最后,代码调用启动屏幕对话框具有 `delay` 参数值的 `showSplash` 方法。该代码几乎与我们在第 4 章添加应用程序启动过程中显示启动屏幕的支持时添加到 WFCJot 应用程序中的代码完全相同。

第二个方法 `showFindAndReplace` 将 `com.ms.wfc.ui.Form` 对象和 `Edit` 控件对象的引用作为参数接受。该方法的代码创建 `FindAndReplace` 对话框的实例,然后调用 `FindAndReplace` 对话框的 `setParent` 方法,并将 `parent` 参数的值传递给方法。这将使对话框能够绑定到在调用中传递到方法的父窗口中。为了绑定 `FindAndReplace` 对话框要使用的 `Edit` 控件,代码将调用 `setEditControl` 方法,并将它传递给 `edit` 参数。

现在我们已经正确地将对话框绑定并使它成为父窗口,代码最后将调用 `show` 方法。这将使该对话框成为指定窗口和 `Edit` 控件的父窗口,但这不能实际显示出来。下一步,编译工程,以便在“注册表”中注册 COM 组件,并将它打包到 COM DLL 中。

13.4.4 编译 COM 客户应用程序

现在已经完成并编译了 COM 组件,下面需要创建客户应用程序来测试它的功能。在这一步骤中,将创建一个简单的使用该 COM 组件的客户应用程序。通过从 **File** 菜单中选择 **Add Project**,将一个工程添加到当前的解决方案中。确保选中 **Add To Current Solution** 选项,然后使用 **Empty Project** 模板创建一个新的工程,并将其命名为 `FormReuseClient`。**Visual J++** 将把该工程添加到 **Project Explorer** 及当前的解决方案中。记住,解决方案只不过是相关工程的集合。

将一个窗体添加到工程中,并将其命名为 `ClientForm.java`。将窗体的 `text` 属性设置为 `Form Reuse Client`。将 `Edit` 控件添加到窗体中,将其命名为 `txtDocument`,并清除其文本属性;添加一个 `ContextMenu` 控件,并将其命名为 `cmnClient`。将 `txtDocument` 的 `dock` 属性设置为 `Fill`,将 `multiline` 属性设置为 `true`,将 `scrollBars` 属性设置为 `Vertical`,将 `hideSelection` 属性设置为 `false`。要将环境菜单与 `Edit` 控件联系起来,可以通过从列表中选择来将 `Edit` 控件的 `contextMenu` 属性设置为 `cmnClient`。在 `cmnClient` 控件中,添加一个菜单项,然后将它命名为 `mnuFindAndReplace`。将该菜单项的 `text` 属性设置为 `Find And Replace Text`。

13.4.5 导入 FormReuse COM 组件

设计完窗体,下面我们可以将 `FormReuse` COM 组件导入到应用程序中。在

Project Explorer 中选定客户工程,显示 COM Wrappers 对话框,然后从列表中选择 FormReuse COM 组件,单击 OK 按钮。Visual J++ 为组件创建完 COM 封装程序后,将下面的代码添加到 ClientForm 类的前面,来创建 COM 组件的私有实例:

```
private FormReuse dialogs = new FormReuse();
```

现在已经导入 COM 组件,并在 ClientForm 类中定义了它的实例,下面可以将支持代码添加到客户应用程序中。为 mnuFindAndReplace 菜单项的 click 事件创建一个处理程序。将下面的代码添加到事件处理程序中:

```
dialogs.showFindAndReplace(this, txtDocument);
```

这行代码使用了 COM 组件的私有实例 dialogs,并调用了它的 showFindAndReplace 方法。通过传递调用应用程序窗体和 txtDocument Edit 控件引用的方法。

为了完成客户代码,将下面的代码添加到 ClientForm 构造函数所有其他代码的后面:



```
dialogs.showSplash("Mountain Valley Software",
    "Logo.bmp",
    "Clouds.bmp",
    "Copyright 1998, My Virtual Company",
    "This is warning text.",
    "Virtual User",
    "My Application",
    "Version 1.0a",
    5000);
```

这几行代码调用 showSplash 方法,并传递要在启动屏幕上显示的版权、版本、产品、警告文本、用户及公司等信息。showSplash 调用还将传递两个位图参数(一个用于徽标,另一个用于背景)及延迟关闭启动屏幕的微秒数。

将本书附带光盘中的 clouds.bmp 和 logo.bmp 位图添加到工程中。当您要在系统中执行该应用程序时,或许需要更改位图的路径,因为它可能无法在您的工程目录中找到这些位图。这是因为在 Visual J++ 中运行工程时,默认的路径是 WJView.exe 应用程序所在的 Windows 目录。Visual J++ 使用 WJView 运行应用程序并将在 Windows 中显示它。如果您是通过可执行文件(.exe)来执行应用程序,并且位图保存在那个目录中,那么它们可以得到定位,因为.exe 是实例应用程序的文件,而不是 Visual J++。

现在已经完成客户应用程序。因为我们使原来的对话框和 COM 组件非常易于操纵,所以我们的客户应用程序只需要少量的代码。编译工程,在运行它之前,右击 FormReuseClient 工程,然后选定 Set As Startup Project。通知 Visual J++ 在

运行解决方案时将使用该工程。

在运行应用程序时,将会与我们的老朋友——启动屏幕打个招呼。它将像在第8章中那样显示有徽标和背景位图。启动屏幕关闭并显示应用程序窗体后,将一些文本输入到 Edit 控件中。然后右击 Edit 控件,选择 Find And Replace Text 菜单项。这时 FindAndReplace 对话框将显示为无模式窗口。

正如我们在该实例中所看到的,可以将任意类型的可重用用户接口合并到 COM 组件中。下面是一些可以添加到用户接口 COM 组件中的常见对话框。

- ◆ About 框。
- ◆ 长文件或通信进程的进度窗口。
- ◆ 目录选定内容对话框。
- ◆ 数据库搜索窗口(在下面的实例中,我们将创建一个)。
- ◆ 能够向客户服务部提供有用数据的系统信息窗口。

13.5 数据库 COM 组件

几乎所有开发的应用程序都会需要访问某些类型的数据库。无论您是为 mom and pop 小杂货店创建一个维护库存和费用的应用程序,还是为一家航空公司设计三维模拟环境,都需要访问数据库。并且经常会发现需要自己编写代码来验证输入的数据,或编排从数据库读取的数据的格式。或许还会发现,应用程序需要向用户提供从数据库选择指定记录或搜索记录组的方法。所有这些任务都可以由 COM 组件来很好地完成。

比如有这样一种情况:需要开发一套处理库存、订单等信息的应用程序。每一个应用程序都有一个独立的可执行文件。该套件中几乎每一个应用程序都常常需要具有搜索库存工程或客户记录的能力。为了使每个应用程序能够共享搜索和返回记录的公共用户接口,需要创建一些 COM 组件,使用户能够在一致的用户接口中搜索记录,并以相同的形式将记录返回到客户应用程序中。

在该实例中,将创建一个与数据库交互的 COM 组件。它将向用户显示一个对话框,并允许他们在 Northwind 数据库的 Customers 表中搜索指定字段。然后他们可以在要使用的 DataGrid 控件中选择客户记录,并单击 OK 按钮。COM 组件将返回 DataSource 对象,客户应用程序可以用它来读取和修改指定的客户记录。图 13.6 显示了 COM 组件的对话框在运行时的外观。

在第 14 章“Visual J++ 应用程序的高级实例”中,我们将在图书订购条目应用程序中使用类似的客户搜索引擎。完成该实例后,您将可以在自己的数据库

应用程序中使用作为创建类似数据库驱动的 COM 组件基础的代码。

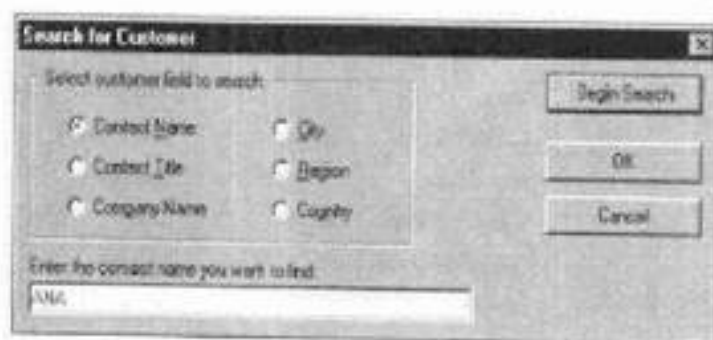


图 13.6 运行时的 CustomerSearchEngine COM 工程

13.5.1 创建 COM 工程

首先创建一个新的 COM DLL 工程, 将它命名为 CustomerSearchEngine。重新命名该工程添加到 CustomerSearch.java 的类文件中, 并用 CustomerSearch 替换源文件中所有的 Class1 实例。我们将使用 COM 类来提供 COM 客户和搜索对话框之间的提取层。将一个新窗体添加到工程中, 并将其命名为 CustomerSearchDlg.java。

13.5.2 设计 CustomerSearchDlg

在 Forms Designer 中打开 CustomerSearchDlg.java 文件。使用图 13.7 在窗体中适当地排列控件。使用表 13.3 将控件添加到窗体中, 并设置它们的属性。

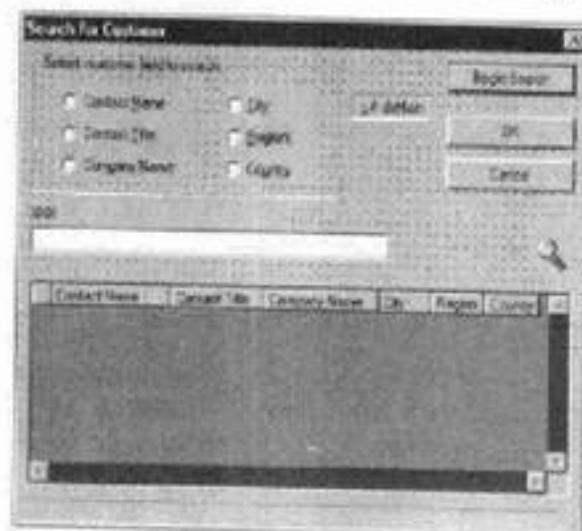


图 13.7 设计时的 CustomerSearchDlg.java

表 13.3 CustomerSearchDlg.java 组件和属性设置

组件类型	属性	设置
Form	text	Search for Customer...
	borderStyle	Fixed Dialog
	cancelButton	btnCancel
	minimizeButton	false
	maximizeButton	false
	startPosition	Center Screen
GroupBox	name	grpSearchType
	text	Select customer field to search:
RadioButton	name	optContactName
	text	Contact &Name
	tabIndex	0
	checked	true
RadioButton	name	optContactTitle
	text	Contact &Title
	tabIndex	1
RadioButton	name	optCompanyName
	text	Com&pary Name
	tabIndex	2
RadioButton	name	optCity
	text	&City
	TabIndex	3
RadioButton	name	optRegion
	text	&Region
	tabIndex	4
RadioButton	name	optCountry
	text	Co&untry
tabIndex	5	
Label	name	lblSearchVal
	text	XXX
	tabIndex	1
Edit	name	txtSearchVal
	text	" "
	TabIndex	2
DataSource	Name	dsMain
	lockType	Optimistic
	mode	ReadWrite
	designTimeData	false

续表

组件类型	属性	设置
DataGrid	name	cbgResults
	tabIndex	6
	visible	false
	allowAddNew	false
	allowDelete	false
	AllowUpdate	false
	dataSource	dsMain
	dynamicColumns	false
Animation	name	annSearch
	visible	false
	transparent	true
StatusBar	name	statSearch
	showPanels	true
	visible	false
Button	name	btnSearch
	text	Begin Search
	tabIndex	3
Button	name	btnOK
	enabled	false
	text	OK
	dialogResult	OK
	tabIndex	4
Button	name	btnCancel
	text	Cancel
	dialogResult	Cancel
	tabIndex	5

为了完成 CustomerSearchDlg 用户接口的设计,需要配置 DataSource 控件,将窗格添加到 StatusBar 控件中,然后配置 DataGrid 控件的列。首先我们来配置 DataSource 控件。在本书附带的光盘中,包括了 Northwind.mdb 数据库的副本。CustomerSearchEngine COM 组件将使用这个数据库的 Customers 表来搜索客户记录。

单击 DataSource 控件 ConnectionString 属性值字段中的 Ellipsis 按钮。在 Data Link Properties 对话框的 Provider 选项卡中,选定 Microsoft Jet 3.51 OLE DB Provider。在 Connection 选项卡中,定位到系统中的 Northwind.mdb 文件。单击 OK 按钮,将更改提交到 DataSource 控件中。因为我们将运行时在对话框中创建查询,所以现在保留 commandText 属性为空。回顾第 11 章“数据库编程”中我们对数据库的讨论,在本书附带光盘的实例中,在所有拥有 DataSource 对象的窗体

构造函数中都包括了注释外代码。如果您希望将这个实例发布到其他系统中,那么可以使用该代码来将属性与数据库联系起来。为了使用代码,将 Northwind.mdb 文件移动到在 `setConnectionString` 方法调用中指定的目录下。请记住, `getSpecialFolderPath` 调用通常返回 `\Windows\Application Data` 文件夹。

接下来定义 `StatusBar` 控件的窗格,我们将用它来显示在搜索会话中找到的记录数。单击 `StatusBar` 控件 `panels` 属性值区的 `Ellipsis` 按钮。在 `StatusBarPanel` 编辑器中,单击 `Add` 按钮,然后单击 `OK` 按钮。将窗格的 `autoSize` 属性设置为 `Spring`,将它的名称属性设置为 `stpMain`,并清除它的 `text` 属性。

最后需要定义 `DataGrid` 控件的列。在 `Forms Designer` 中选定 `DataGrid`,然后单击 `columns` 属性值区的 `Ellipsis` 按钮。在 `Column Editor` 对话框中,添加 6 个列,然后单击 `OK` 按钮。使用表 13.4 来定义刚刚添加的每个列的属性。

表 13.4 CustomerSearchDlg.java `DataGrid` 列属性

列索引	属性	设置
0	<code>name</code>	<code>ColContactName</code>
	<code>boundFieldName</code>	<code>ContactName</code>
	<code>caption</code>	<code>Contact Name</code>
	<code>readOnly</code>	<code>True</code>
1	<code>name</code>	<code>ColContactTitle</code>
	<code>boundFieldName</code>	<code>ContactTitle</code>
	<code>caption</code>	<code>Contact Title</code>
	<code>readOnly</code>	<code>True</code>
2	<code>name</code>	<code>colCompanyName</code>
	<code>boundFieldName</code>	<code>CompanyName</code>
	<code>caption</code>	<code>Company Name</code>
	<code>readOnly</code>	<code>true</code>
3	<code>name</code>	<code>colCity</code>
	<code>boundFieldName</code>	<code>City</code>
	<code>caption</code>	<code>City</code>
	<code>readOnly</code>	<code>true</code>
4	<code>name</code>	<code>colRegion</code>
	<code>boundFieldName</code>	<code>Region</code>
	<code>caption</code>	<code>Region</code>
	<code>readOnly</code>	<code>true</code>
5	<code>name</code>	<code>colCountry</code>
	<code>boundFieldName</code>	<code>Country</code>
	<code>caption</code>	<code>Country</code>
	<code>readOnly</code>	<code>true</code>

13.5.3 将支持代码添加到 CustomerSearchDlg 中

现在我们已经设计了 CustomerSearchDlg.java 对话框的用户接口,下面我们来将支持代码添加到对话框中。首先需要添加一些导入语句和一些在整个对话框中使用的私有成员变量。将下面的导入语句添加到 CustomerSearchDlg 类的源文件前面:

```
import com.ms.wfc.util.*;
import com.ms.wfc.data.ui.*;
```

com.ms.wfc.data.ui 数据包的导入语句将允许在代码中访问 DataSource、DataGrid 和 Column 对象的方法。com.ms.wfc.util 数据包向代码提供了使用在字符串两端添加引号的静态方法 Utils.quoteOfString 的能力。在定义客户搜索的查询时,将使用这个方法。

下面我们来添加对话框源代码将要使用的私有成员变量。将下面的代码添加到 CustomerSearchDlg 类定义的前面:



```
private String msAppName = "Customer Search";
private final int SEARCHMODE = 0;
private final int STARTMODE = 1;
private final int STARTSIZE = 190;
private final int SEARCHSIZE = 380;
private int miOpMode = STARTMODE;
private DataSource mdsSelCustomer;
```

对话框中的消息框使用 msAppName String 变量。SEARCHMODE 和 STARTMODE 整型变量定义对话框的模式。这将使代码能够确定显示用户接口的方式。miOpMode 变量存储对话框当前操作的状态。STARTSIZE 和 SEARCHSIZE 整型变量定义对话框的高度。

首次显示该对话框时,将在视图中隐藏 DataGrid、Animation 和 StatusBar 控件,以减小对话框的大小。当对话框开始搜索客户记录时,它将增大对话框来显示 DataGrid 和 StatusBar 控件。mdsSelCustomer 变量将存储选定的客户记录。COM 组件将把 mdsSelCustomer 中的 DataSource 值返回到客户应用程序中。

将下面的代码添加到 CustomerSearchDlg 类中:



```
/* * Handles setting the search entry caption */
private void ChangeSearchCaption()
{
    if (optContactName.getChecked() == true)
```

```

        lblSearchVal.setText ("Enter the contact name "
                               + "you want to find:");
    else if(optContactTitle.getChecked() == true)
        lblSearchVal.setText ("Enter the contact title "
                               + "you want to find:");
    else if(optCompanyName.getChecked() == true)
        lblSearchVal.setText ("Enter the company name "
                               + "you want to find:");
    else if(optCity.getChecked() == true)
        lblSearchVal.setText ("Enter the city "
                               + "you want to find:");
    else if(optRegion.getChecked() == true)
        lblSearchVal.setText ("Enter the region or state "
                               + "you want to find:");
    else
        lblSearchVal.setText ("Enter the country "
                               + "you want to find:");
}
/* * Determines which option is selected, and returns a string with
 * the field name to search for
 */
private String GetSearchField()
{
    if (optContactName.getChecked() == true)
        return "ContactName";
    else if(optContactTitle.getChecked() == true)
        return "ContactTitle";
    else if(optCompanyName.getChecked() == true)
        return "CompanyName";
    else if(optCity.getChecked() == true)
        return "City";
    else if(optRegion.getChecked() == true)
        return "Region";
    else
        return "Country";
}
/* * Click event handler for the option buttons. Updates label for
 * entry value.
 */
private void ChangeSearchOption(Object source, Event e)
{
    // Update the label control based on the current search field
    ChangeSearchCaption();
}

```



```

/* * Click event handler for the btnBeginSearch button.
 * Either starts or stops a search for a record.
 */
private void BeginSearch(Object source, Event e)
{
    // Check the mode of the dialog box
    if(miOpMode == STARTMODE)
    {
        /* Determine whether the user
         * failed to enter a value to search for */
        if (txtSearchVal.getText().equals(""))
        {
            // Alert user that a search value was not entered
            MessageBox.show("Please enter a value to search for",
                            msAppName,
                            MessageBox.ICONEXCLAMATION);

            // Set the focus to the Edit control
            this.setActiveControl(txtSearchVal);
            return;
        }

        // Start the search animation, and make it visible
        annSearch.setVisible(true);
        annSearch.setAutoplay(true);
        // Set the height of the dialog box to its search height
        this.setHeight(SEARCHSIZE);

        // Make the DataGrid and StatusBar controls visible
        dbgResults.setVisible(true);
        statSearch.setVisible(true);
        // Change the mode of the dialog box to search mode
        miOpMode = SEARCHMODE;

        // Set the search button to be the stop button
        btnSearch.setText("Stop");
        /* Call SearchForRecord to do the actual search.
         * Store the return value. */
        int numRecs = SearchForRecords();
        /* If there were no records returned,
         * alert the user and reset the button */
        if ( numRecs == 0)
        {
            MessageBox.show("No matching records were found!",
                            msAppName,
                            MessageBox.ICONEXCLAMATION);
        }
    }
}

```

```

else
|
|   /* Change the statusbar text to indicate
|   * the number of records that matched */
|   statSearch.getStatusBarPanel(0)
|       .setText(Integer.toString(numRecs)
|           + " matching records were found."
|       );
|   /* Reset the search button text for the next search
|   operation */
|
|   // Stop the animation, and make it invisible
|   annSearch.setAutoplay(false);
|   annSearch.setVisible(false);
|
|   // Reset the mode of the dialog box
|   miOpMode = STARTMODE;
|   // Reset the search button
|   btnSearch.setText("Begin Search");
|
| /* * Helper method that does the actual search in the database.
| * Returns an integer that indicates the number of records
| * found that matched the criteria entered into
| * the txtSearchVal Edit control.
| */
private int SearchForRecords()
|
|   int returnVal = 0;
|   // Get the string to search for
|   String searchField = GetSearchField();
|   // Append the SQL wildcard character to the value
|   String searchValue = txtSearchVal.getText() + "%";
|   // Place quotes around the string
|   searchValue = Utils.quotedString(searchValue);
|
|   // Set the data source's query
|   dsMain.setCommandText("Select * FROM Customers WHERE "
|       + searchField
|       + " Like " + searchValue);
|   // Disable the btnOK button until we have valid records
|   btnOK.setEnabled(false);
|
|   // Check to see whether we have a valid recordset object
|   if (dsMain.getRecordset() != null)
|   |
|   |   /* To ensure that we have a valid recordset,

```

```

        * check for end and beginning of file */
        if (! dsMain.getRecordset().getBOF()
            && ! dsMain.getRecordset().getBOP())
        {
            // Enable the OK button so that the user can continue
            btnOK.setEnabled(true);
            /* Return the number of records
             * that were found that matched */
            returnVal = dsMain.getRecordset().getRecordCount();
        }
    }
    return returnVal;
}

/* * Click event handler for the OK button. Stores information
 * so that CustomerSearch can obtain it.
 */
private void StoreRecord(Object source, Event e)
{
    // Store all of the selected items in the grid
    Object[] tempSel = dbgResults.getSelectedRows();
    Object curCustomer;

    /* If the selection array is not empty, select first element
    in array as the selected customer */
    if (tempSel.length != 0)
    {
        curCustomer = tempSel[0];
    }
    else
    {
        /* The user probably selected a cell and not the entire row.
        Use the current row instead. */
        curCustomer = dbgResults.getCurrentRow();
    }

    /* Now that we have the bookmark to the current customer,
    * we need to create a recordset for it */

    // Move to the customer record that was selected
    dsMain.getRecordset().move(0, curCustomer);
    /* Create the SQL statement to pass back
    * to the Datasource control */
    String customerID = Utils
        .quotedString(dsMain
            .getRecordset()
            .getField("CustomerID"))

```

```

        .getString());

String SQLString = "Select * FROM Customers"
                  + "WHERE CustomerID = " + customerID;
/* Assign the query that creates a recordset with
 * the selected customer record */
dsMain.setCommandText(SQLString);
/* Store the data source for the recordset in the
 * private member variable */
mdsSelCustomer = dsMain;
|
|
/* * Returns the database bookmark to the selected customer
 * from the dialog box */
public DataSource getSelectedCustomer()
|
|
    return mdsSelCustomer;
|
|
/* * Disposes the dialog box
 * */
private void ExitDialog(Object source, Event e)
|
|
    dispose();
|
|

```

在研究添加的代码之前,先来将事件处理程序与它们的事件源联系起来。我们需要将 ChangeSearchOption 事件处理程序与窗体中的所有 6 个按钮的 click 事件联系起来,并要将 BeginSearch 事件处理程序与 btnSearch Button 控件的 click 事件联系起来。将 StoreRecord 事件处理程序与 btnOK Button 控件的 click 事件联系起来。最后,将 ExitDialog 事件处理程序与 btnCancel Button 控件的 click 事件联系起来。

ChangeSearchCaption。下面,方法将更改 Edit 控件 lblSearchVal 上 Label 控件的标题。当选中选项按钮时,lblSearchVal 控件的文本属性都会改变来反映用户希望输入的值的类型。这就是在控件的 text 属性中输入实际值的原因。代码经过一个 if/else 块来确定当前选定的 RadioButton 按钮控件。当它找到当前控件后,它将根据选中的选项设置 Label 控件的 text 属性。单击 RadioButton 时触发的 ChangeSearchOption 事件处理程序将调用这个方法作为它独立的代码。

GetSearchField 方法确定数据库中哪个字段将作为搜索字段。GetSearchField 方法的代码类似于 ChangeSearchCaption 方法的代码,它根据代码 if/else 块确定选中的 RadioButton。确定当前选中的 RadioButton 后,它将返回表示搜索字段名称的文本。

当用户单击 btnSearch Button 控件时将触发 BeginSearch 事件处理程序,并负

责启动客户搜索进程。BeginSearch 的代码检查对话框当前的可操作状态,它是通过确定 miOpMode 成员是否被设置为 STARTMODE 来做到这一点的。如果模式被设置为 STARTMODE,那么代码将启动搜索进程。否则,代码重新把对话框的模式设置为 STARTMODE,将 btnSearch 的标题重新设置为 Begin Search,然后退出事件处理程序。当搜索进程结束时还将调用该代码,以为下一个搜索操作做准备。

当 miOpMode 等于 STARTMODE 时执行的代码从检查 txtSearchVal Edit 控件的值来搜索数据库开始。如果值为空,那么代码将显示一个消息框,将 txtSearchVal 设置为活动控件,然后退出事件处理程序。如果将有效的搜索值输入到 txtSearchVal 中,那么代码将显示 Animation 按钮。代码还将通过调用 Animation 控件作为参数的具有 true 值的 setAutoPlay 方法来启动播放动画。如果您具有一个相对较快的系统,那么可能会看不到动画,因为搜索过程很快。只有当数据库很大或者数据库处于远程服务器上时动画才会有用。

启动动画后,代码将继续设置窗体的高度,以使 DataGrid 和 StatusBar 控件能够显示。将 btnSearch 的 text 属性设置为 Stop 后,代码将调用 SearchForRecords 方法。该方法将执行实际的客户记录搜索。我们将在下一节中讨论这个问题。SearchForRecords 将返回找到的记录数。该值将以本地整型值的形式存储,并用于下面的代码来确定找到的记录是否与搜索准则匹配。如果没有找到匹配的记录,那么将会出现一个消息框通知用户。否则,StatusBar 控件的窗格 statSearch 将更新以显示搜索找到的记录数。STARTMODE 的代码最后终止 Animation 控件,并使它消失。

正如前面提到的,SearchForRecords 方法执行实际的记录搜索工作。在用户通过 RadioButtons 选定的字段中,它将在数据库中查询与输入到 Edit 控件中的文本匹配的记录。该方法的代码从创建存储与准则匹配的记录数的整型变量开始。该值将用作方法的返回值。

接下来代码将创建 String 对象 searchField 的实例,并使用前面讨论过的 GetSearchField 方法的值来初始化它。一旦 GetSearchField 返回 SearchForRecords 将要用来自数据库的字段时,代码就将创建另一个 String 对象 searchValue,它将被用来存储 txtSearchValEdit 控件 text 属性的值。代码将标准的 SQL 通配符(*)附加到 Edit 控件的文本中。因为我们无法确定用户是否输入完整的搜索字段值,所以 SearchForRecords 将使用 LIKE 运算符来代替等号来执行所有的 SQL 查询,并且它通常使用包含通配符的搜索值。

接下来,代码将在 searchValue String 对象值两端加上引号,因为要在查询中使用这个 String 对象,根据 SQL 语句中的准则,必须包括在引号中。为了在字符串两端加引号,我们使用了 com.ms.wfc.util.Util.quoteString 静态方法。

设置完搜索字段和搜索值后,代码将把查询分配到 DataSource 控件 dsMain 的 commandText 属性。传递的 SQL 查询使用我们前面定义的 searchField 和 searchValue String 对象。将查询分配给 dsMain 后,代码将禁用 btnOK Button 控件,因为当前没有选择并查看 DataSource 控件中的记录集是否等于 null 的记录。如果没有,那么代码将通过查找等于 End Of File(EOF)和 Beginning Of File (BOF)标记的记录集来两次检查是否存在有效的记录集。

回顾第 11 章“数据库编程”中所讨论的,通过查看 BOF 标记和 EOF 标记是否是相同的记录来执行这种检查以确保记录集中有记录。如果 BOF 标记和 EOF 标记是相同的记录,那么这表示它是一个空记录集。如果其中的一个条件当前没有满足,那么方法的代码最后将启用 btnOK Button 控件(因为我们现在已经有可以在 DataGrid 控件中选择的记录),然后设置 returnVal 等于 DataSource 控件记录集中的记录数。该值作为方法的返回值。当该代码结束时,绑定到 dsMain 的 DataGrid 控件将使用 dsMain 记录集中的记录填写它的行。

对话框最后一个主要方法是 StoreRecord。当用户单击 OK 按钮时触发该事件处理程序,它负责减少 DataSource 控件的记录集,以使它只包含用户在 DataGrid 控件中选定的记录。该方法的代码创建类型 Object 的阵列,并使用 DataGrid 控件的 getSelectedRows 方法调用的返回值初始化它。该方法返回表示 DataGrid 控件中当前所选行的数据库书签的阵列。

在实例中,我们只关心选定的第一个记录。我们还将初始化为 curCustomer 的对象,将用它来存储当前所选客户记录的书签。如果阵列不是空的,那么代码将把阵列中的第一个元素存储到 curCustomer 对象中。否则,如果因为某些原因阵列是空的,那么代码将把 curCustomer 指定为 getCurrentRow 的返回值,它将返回光标所在的 DataGrid 控件中的当前行。

存储所选客户记录的书签后,代码将调用 dsMain 的 getRecordset 方法来获得记录集的引用。在相同的 getRecordset 调用中,代码将调用 move 方法,并将书签传递给存储在 curCustomer 中的所选客户记录。move 方法将 dsMain 的记录集中的光标移动到用户选定的记录中。调用 move 方法后,代码将提取数据库 CustomerID 字段的值,并将它存储在名为 customerId 的 String 对象中。

CustomerID 字段是唯一的索引,所以它存储在 customerId String 对象中,以便用户选定的记录可以在方法的代码中很容易地得到检索。像 SearchForRecords 方法中的搜索值一样,customerId 的值具有索引号,以使它能够用于 SQL 查询中。代码然后创建一个名为 SQLString 的 String 对象,其中包含用来获得用户请求的特定记录的 SQL 完整查询。该 String 对象然后被传递给 dsMain 的 commandText 属性来执行查询。DataSource 控件的记录集将只包含用户从 DataGrid 控件选定的一个记录。代码最后将 dsMain 存储在前面定义的私有成员变量 mdsSel-

Customer 中。

最后,要看的几个方法很简单。getSelectedCustomer 方法被 CustomerSearch 类调用,并且它只返回包含 COM 组件用户请求的记录的 DataSource 对象。btnCancel 的 click 事件触发 ExitDialog 事件处理程序。与大多数“现有的”事件处理程序不同,这个事件处理程序调用 dispose,而不调用 Application.exit。原因是我们不想退出应用程序,而只是退出对话框。如果在 COM 组件中调用 Application.exit,那么需要之前终止 COM 组件。

为了完成对话框,我们添加的最后一些代码将修改构造函数。代码将 lblSearchVal Label 控件的标题设置它的初值,并设置对话框的高度。将下面的代码添加到 CustomerSearchDlg 构造函数所有代码的后面:



```
// Initialize the size of the dialog box
this.setHeight(STARTSIZE);
// Initialize the caption for the label above the search value
ChangeSearchCaption();
// Specify the location for the animation file
animSearch.setFileName(SystemInformation
    .getSpecialFolderPath(SpecialFolder.APPLICATIONDATA) +
    "\\DeveloperWorkshop\\CustomerSearchEngine\\FindFile.
    avi");
```

13.5.4 将代码添加到 CustomerSearch COM 类中

为了完成 COM 组件,需要将代码添加到通过 COM 展示的 CustomerSearch 类中。在添加方法之前,需要将一些导入语句添加到 CustomerSearch.java 源文件之前。添加下面的导入语句:

```
import com.ms.wfc.data.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.data.ui.*;
```

将下面的代码添加到 CustomerSearch 类中:



```
private CustomerSearchDlg csDlg = new CustomerSearchDlg();
private DataSource mdsSelCustomer;
/** Displays a search dialog box for retrieving a customer record
 * /
public void showCustomerSearchDialog()
{
    // Show the Customer Search Engine dialog box
```



```

        csDlg.ShowDialog();
        /* If the user selected the OK button, retrieve the
         * data source for the selected customer record */
        if (csDlg.DialogResult() == DialogResult.OK)
        {
            /* Store the data source for the selected
             * customer record in a private member variable */
            mdsSelCustomer = csDlg.GetSelectedCustomer();
        }
    }
    /** Returns a data source that contains
     * the single customer record that is selected
     */
    public IDataSource GetSelectedCustomer()
    {
        return mdsSelCustomer;
    }
}

```

首先添加两个私有成员变量:csDlg(CustomerSearchDlg 类的实例)和 mdsSelCustomer(COM 组件用来存储在 CustomerSearchDlg 对话框中定义的 DataSource 的对象)。

添加的第一个方法是 showCustomerSearchDialog。该方法通过 COM 展示。它将显示 CustomerSearchDlg 对话框,并检索包含所选客户记录的 DataSource 对象,然后将它存储到本地的 DataSource 对象 mdsSelCustomer 中。

要添加的最后一个方法是 getSelectedCustomer。在应用程序通过调用 showCustomerSearchDialog 显示对话框,并在用户做出选择后,客户应用程序将调用 getSelectedCustomer 方法来检索包含客户记录的 DataSource 对象。getSelectedCustomer 方法的返回值为 IDataSource 接口而不是 DataSource,COM 组件允许支持 IDataSource 接口的语言来检索值。

现在已经完成了 COM 组件的源代码。编译工程。在下一步骤中,我们将把一个新工程添加到作为 CustomerSearchEngine COM 组件客户的解决方案中。

13.5.5 编译 CustomerSearchEngine 客户应用程序

现在已经完成了 COM 组件的设计工作,下面需要使用应用程序来测试它的功能。打开 CustomerSearchEngine,从 File 菜单中选择 Add Project。在 Add Project 对话框中,确保选中 Add To Current Solution 选项,然后使用 Empty Project 模板创建一个新的工程。将该工程命名为 CustomerSearchEngineClient。将一个新窗体添

加到工程中,并将它命名为 CustomerClient.java。在代码视图中,将下面的导入语句添加到 CustomerClient.java 文件的开始处,其他导入语句之后:

```
import com.ms.wfc.data.*;
import com.ms.wfc.data.ui.*;
```

在 Forms Designer 中打开 CustomerClient,然后使用表 13.5 来将控件添加到窗体中,并指定它们的属性。

表 13.5 CustomerClient.java 组件及属性设置

组件类型	属性	设置
Form	text	Customer Engine Search Client
	borderStyle	Fixed Dialog
	maximizeButton	false
Label	name	lblCustomerName
	text	Customer Name:
	tabIndex	0
Edit	name	txtCustomerName
	text	" "
	tabIndex	1
Label	name	lblCustomerTitle
	text	Customer Title:
	tabIndex	2
Edit	name	txtCustomerTitle
	text	" "
	tabIndex	3
Label	name	lblCompany
	text	Customer's Company:
	tabIndex	4
Edit	name	txtCompany
	text	" "
	tabIndex	5
DataSource	name	dsMain
	designTimeData	false
Button	name	btnSearch
	text	Search
	tabIndex	6
Button	name	btnClose
	text	Close
	tabIndex	7


```

        + "customer information!",
        "Customer Search Client",
        MessageBox.ICONERROR);

    else

        MessageBox.show("ERROR: The Customer Search Engine "
            + "was not able to provide "
            + "customer information!",
            "Customer Search Client",
            MessageBox.ICONERROR);

private void ExitApp(Object source, Event e)

    Application.exit();

```

SearchForCustomer 事件处理程序的代码创建 CustomerSearchEngine COM 组件的实例。然后调用组件的 showCustomerSearchDialog 方法来显示 Search For Customer 对话框。当用户从 DataGrid 控件中选定客户记录并单击 OK 后,客户应用程序将重新获得控件,并调用 COM 组件的 getSelectedCustomer 方法。getSelectedCustomer 的返回值被分配给客户应用程序的 DataSource 控件。

如果因为某种原因,DataSource 控件为空,那么将会出现一个消息框通知用户。如果 DataSource 控件不为空,那么代码将创建一个临时的记录集,并使用 DataSource 控件中当前存储的记录集来初始化它。然后代码检查记录集,以确保它不为空。如果它不为空,那么代码将检索被检索记录 ContactName、ContactTitle 和 CompanyName 字段的值。

现在我们准备编译和运行工程。在 Project Explorer 中右击,然后选择 Set As Startup Project,使 CustomerSearchEngineClient 成为活动工程。另外将文件 Findfile .uvi 和 Northwind.mdb 从本书附带光盘的 CustomerSearchEngineClient 文件夹复制到新建的 CustomerSearchEngineClient 文件夹中。编译并运行工程。当显示客户应用程序的窗体时,单击对话框中的 Search 按钮,显示 Search For Customer 对话框。选中 Contact Name 按钮,然后输入搜索值 Ana。单击 Begin Search 按钮,您将从数据库中检索到两个记录。在 DataGrid 中选定最后一个,然后单击 OK 按钮。这时指定的记录将显示在客户应用程序中。

这个 CustomerSearchEngine COM 组件非常有用,它使您在应用程序中创建类似的基于 COM 的搜索引擎方面有了一个好的开端。以下是如何通过包括更多的功能来增强这个 COM 组件的几点建议。

◆ 代替使用固定字段,向用户提供指定要搜索的字段的方法,并顺便为这些字

段创建单选按钮。这将需要进行更多的工作,因为您必须限制可以选择的搜索字段数目,或者动态地调整对话框 GroupBox 控件的大小。

- ◆ 为 COM 组件提供一种方法,使它能够动态地了解有关它正在使用的数据库的信息。提供一些属性和方法,使用户能够在运行时指定数据库。
- ◆ 向用户提供一种搜索多个字段和使用多个搜索值的方法。例如,搜索引擎的用户或许希望通过提供除原来的字段和值之外的其他要搜索的字段和值,来缩小搜索范围。

13.6 第三方 COM 组件

COM 一个最有价值的功能是,它可以合并通过 COM 从其他应用程序展示到您的应用程序中的功能和特性。很多公司通过 COM 展示了它们整个应用程序接口,像 Microsoft 的 Microsoft Excel、Microsoft Word,甚至 Visual J++ 6.0! 您的程序可能使用 Microsoft Excel 中的数学函数,或者创建和打印 Microsoft Access 的数据库报表。

可以合并很多在第三方应用程序中通过 COM 展示的功能,来代替在您的代码中设计相同的功能。通过重复使用其他应用程序的功能,可以减少开发应用程序所需的时间。例如,使用 Microsoft Excel 和 Word 来将应用程序中的数据导出到 Microsoft Word 文档或 Microsoft Excel 工作表中。COM 的能力是无穷的!

在下面的实例中,我们将介绍如何在一个简单的 WFC 客户应用程序的文本中使用 Microsoft Word 的拼写功能。图 13.9 显示了该实例在运行时的外观。



图 13.9 运行时的 Office Spelling 应用程序

该实例只演示了应用程序能够通过第三方 COM 组件实现的部分功能。试图在您的应用程序中实现类似的功能之前,强烈建议您仔细研究一下系统中可用的或可以买到的 COM 组件。代码的可重用真是太好了!

13.6.1 创建 WFC 应用程序工程

使用 Empty Project 模板创建一个新工程,并将其命名为 OfficeSpelling。将一个新窗体添加到这个空工程中,并将其命名为 OfficeSpelling.java。将一个 Edit 控件添加到窗体中,并将其命名为 txtDocument。添加一个 ContextMenu 控件,并将它命名为 cmdDoc。通过从值列表中进行选择,将 txtDocument 的 dock 属性设置为 Fill,将它的 multiline 属性设置为 true,将它的 scrollBars 属性设置为 Vertical,将它的 hideSelection 属性设置为 false,并清除它的 text 属性。在 cmdDoc ContextMenu 控件中,创建一个新的菜单项,并将它命名为 mmuSpellCheck,将它的 text 属性设置为 Spell Check with MS Word。

13.6.2 导入 Microsoft Word COM 组件

现在我们已经完成客户应用程序的用户接口,下面将把 Microsoft Word COM 组件导入到工程,以使我们可能访问 Word 的功能。在 Project Explorer 中右击 OfficeSpelling 工程,然后单击 Add。在显示的子菜单中,选择 Add COM Wrapper。在 COM Wrappers 对话框的 Installed COM Components 列表中,查找并在 Microsoft Word COM 组件旁设置一个复选标记。如果安装的是 Microsoft Office 97,COM 组件的名称为 Microsoft Word 8.0 Object Library。COM 组件的名称也许会根据您所安装的 Microsoft Word 的版本而有所不同,但它应该是类似的。

选定 Word COM 组件后,单击 OK 按钮。Visual J++ 将添加一些数据包来安装 COM 封装程序类。vbext1 数据包包含 Word Visual Basic for Applications 对象库的类封装程序。Mso97 数据包包含所有 Microsoft Office 应用程序的常见元素(如 Office 助手)的 COM 封装程序类。msword8 数据包包含 Microsoft Word 展示的主要功能,并且它也将是我们使用的数据包。

13.6.3 将支持代码添加到应用程序中

现在准备将支持代码添加到应用程序中。首先,将下面的导入语句添加到 OfficeSpelling.java 源文件的前面:

```
import msword8.*;
import com.ms.com.*;
```

导入 msword8 数据包只是为了方便起见。导入它,以便我们在所有访问 msword8 数据包的代码前不需要附加 msword8。因为所有参数和 COM 封装程序方法的返回类型都使用 com.ms.com.Variant 对象,所以我们导入这个数据包以使创建 Variant 对象更加容易。

下面我们将添加应用程序剩余的代码。将下面的代码添加到 OfficeSpelling 类的定义中:



```
private Document word = new Document();
private void SpellCheck(Object source, Event e)
{
    /* Assign the size of the document
    * to a Variant to assign to range */
    Variant docSize = new
        Variant(txtDocument.getText().length());

    // Make MS Word visible
    word.getApplication().setVisible(true);

    /* Assign the text from the control
    * to MS Word's current document */
    word.Range(new Variant(),
        new Variant()).setText(txtDocument.getText());

    /* Call the SpellCheck function of MS Word
    * on entire length of text passed */
    word.Range(new Variant(0), docSize)
        .CheckSpelling(new Variant(),
            new Variant(true),
            new Variant(true),
            new Variant(),
            new Variant(),
            new Variant(),
            new Variant(),
            new Variant(),
            new Variant(),
            new Variant(),
            new Variant(),
            new Variant());

    // Select the entire document in Word
    word.Select();

    /* Assign the selected text in Microsoft Word
```



```

    * to the txtDocument Edit control */
    txtDocument.setText(word.getActiveWindow()
        .getSelection()
        .getText());

    //Clear null character added by Microsoft Word
    txtDocument.setText(
        txtDocument.getText()
        .substring(0, txtDocument.getText().length()-1));
    //Select all of the text in the control to show changes
    txtDocument.selectAll();

    //Clear contents of Microsoft Word's document
    word.getActiveWindow().getSelection().setText("");

    //Make Microsoft Word invisible
    word.getApplication().setVisible(false);
}
private void ClosingApp(Object source, CancelEvent e)
{
    word.getApplication().Quit(new Variant(false),
        new Variant(),
        new Variant());
}

```

在讨论代码之前,我们需要将添加的两个事件处理程序与它们的事件源联系起来。将 SpellCheck 事件处理程序与客户应用程序 ContextMenu 控件中的 mnuSpellCheck 菜单项的 click 事件联系起来;将 ClosingApp 事件处理程序与窗体的 closing 事件联系起来。

下面我们来看一下添加的代码。

首先我们添加的是定义 Microsoft Word Document 类实例的私有成员变量。对于所有 Microsoft Word 引用,我们都将使用 Document 类的这个私有实例。当用户单击独立的环境菜单项 mnuSpellCheck 时,将触发 SpellCheck 代码。该方法将执行应用程序所有的搜索操作。

代码将从创建在 txtDocument 中存储文本长度的 Variant 对象开始。然后,代码获得 Microsoft Word Application 对象的实例。在相同的代码中,代码将调用 Application 对象具有 true 值的 setVisible 方法作为它的参数。这将显示 Microsoft Word。请记住,因为 Microsoft Word 是进程外 COM 组件,要引用它,必须创建包含它的应用程序的实例。我们希望显示 Microsoft Word,以便我们可以在进行拼写检查过程中能够看到它显示的 Spell Checker 对话框。

接下来代码将引用 Document 对象的 Range 对象。Range 对象传递两个空的 Variant 对象。通过指定两个空值,Word 将设置 Range 对象来包含整个文档。在

相同的代码中,将调用 Range 对象的 setText 方法。setText 方法被分配为 txtDocument 的 text 属性的值。整个代码将复制 txtDocument 的所有文本,并将它分配给 Microsoft Word 中当前为空的文档中。

然后代码指定文档的 Range 对象应该是文档的文本。这是通过将 0 分配给第一个参数,表示起始位置;将 docSize Variant 对象作为第二个参数,表示 Range 对象的结束位置。记住,我们定义了 docSize 来存储 txtDocument 中的文本长度。我们使用这个长度来确保我们添加的文本就在当前使用的范围的。在相同的代码中,将调用 Range 对象的 CheckSpelling 方法。该方法打开 Microsoft Word 的拼写检查程序窗口,并启动拼写检查进程。

正如您在 CheckSpelling 调用中所看到的,将传递一些空的 Variant 对象。每一个 Variant 对象都将表示在拼写检查时要使用的自定义词典。包含这些值的两个参数都被定义为 true 值。第一个 true 参数确定拼写检查程序是否忽略大小写。这对我们并不是很重要,所以将它设置为 true。第二个 true 参数确定 Word 是否建议与错误拼写单词的拼写匹配的词。该拼写检查程序功能是非常有用的,所以我们将它设置为 true,以确保它可以使用。在关闭拼写检查程序的窗口之前,将不会返回 CheckSpelling 调用。

代码然后使用 Document 对象的 Select 方法来选定 Word 文档中的所有文本,然后使用选定的内容调用 txtDocument 的 setText 方法。要获得 Word 中当前选定的文本,代码首先要获得 Document 对象的活动窗口,然后获得活动窗口的选定对象,然后,它将通过调用 getText 方法来获得当前选定的文本。一行代码管理将文本从 Word 移动到 txtDocument 控件的整个过程!

Microsoft Word 有一个坏习惯,它总是在通过 COM 检索的文本结尾处返回一个空字符。要从 txtDocument 当前文本中删除这个字符,我们将使用 text 属性的值减去最右边的字符值来重新指定 text 属性。代码最后在 txtDocument 中突出显示所有的文本,清除 Microsoft Word 文档中的所有文本,然后隐藏 Microsoft Word。

当关闭应用程序时将使用另一个事件处理程序 ClosingApp。这个事件处理程序将包含终止 Microsoft Word 的一行代码,并通知它不要保存任何文档。这行代码是可选的,但它只做一些清理工作。当您使用像 Microsoft Word 这样的进程外服务器时,如果当前正在运行应用程序的实例,那么 COM 将试着使用它来代替生成一个独立的实例。通过执行 ClosingApp 事件处理程序中的这行代码,将关闭 Microsoft Word,但不保存对打开文档所做的任何更改。这将包括在运行该应用程序时使用的文档。

现在我们就完成了应用程序,编译并运行它。在应用程序中输入一些文本,并故意错误地拼写一些单词。右击 Edit 控件,然后选择 Spell Check With MS Word。这时将显示 Microsoft Word,并开始检查应用程序中的文本。当完成文本

的拼写检查时,新近拼写检查的文本将在 Edit 控件中得到替换。

如果您对通过 COM 实现的大量功能感兴趣,强烈建议您查找一下关于系统中安装的 COM 组件对象层次的文档。Microsoft Office 提供一个完整的层次图表来帮助理解要使用的对象的有效层。

使用 COM 组件是提高应用程序效率及控制其他应用程序功能的好办法。以下是如何改进该实例的一些建议。

- ◆ 将拼写检查的代码分别合并到第 4 章和第 6 章的 WFCJot 和 MDIJot 应用程序中。该程序将使这些应用程序变得卓尔不群,因为大多数文本编辑器没有提供拼写检查功能。
- ◆ 检查完应用程序的拼写后,您可能会问用户是否希望执行语法检查。这种功能也可以通过 Microsoft Word 的 COM 组件获得。
- ◆ 使用 Microsoft Word 的功能将文档导出为其他格式,并以其他数据格式存储应用程序的文本。



更上一层楼

在本章中,我们介绍了在 Visual J++ 应用程序中使用 COM 的方法。在开始介绍 COM 是什么,以及在的开发计划中可以利用它的哪些特性,我们还创建了第一个展示棒球统计函数的 COM 组件。还看到了如何通过为 BaseballMath COM 组件创建客户应用程序来将 COM 组件导入到 Visual J++ 的工程中。通过 COM 显示用户接口也是很重要的,所以我们创建了显示常见用户接口(如在第 8 章“Java 应用程序编程入门”中创建的启动屏幕和 Find And Replace 对话框)的 COM 服务器。

数据库也是应用程序开发的一个重要部分,并且 COM 是提供诸如数据验证和搜索引擎功能的好方法。为了说明这个问题,我们创建了一个显示查找客户记录的用户接口的 COM 组件。当用户选定客户记录并退出对话框时,客户应用程序将提供一个可以用来访问指定客户记录的 DataSource 控件。

最后我们看到的是如何使用第三方 COM 组件(如 Microsoft Word 等)来将一些附加功能添加到应用程序中。为了说明第三方 COM 组件的用法,我们访问了 Microsoft Word 的拼写检查功能来检查小 WFC 应用程序中的文本拼写。

完成本章的实例后,您应牢牢地掌握在应用程序中使用 COM 的方法。在下一章“Visual J++ 应用程序的高级实例”中,当我们合并一些数据库搜索引擎和其他常见用户接口,来简化订单系统的开发过程时,将进一步研究 COM 组件。COM 在很多方面为您的应用程序提供了令人激动的能力!我们强烈建议您探

讨和尝试一下 COM,以巩固一下从本章中获得的知识。

正如您在该实例和 BaseballMath 实例中所看到的,创建 COM 组件和在 Visual J++ 应用程序中使用它们都是很容易的。虽然我们的实例有些简单,但它说明了创建和导入 COM 组件的基本步骤。可以采用下面的建议来提供更多的功能:

- ◆ 将更多的方法添加到棒球统计 COM 组件中,来提供诸如净胜场次(用来确定一个球队必须赢得多少场才能进入下一轮)、偷垒百分比、出局百分比,以及平均击球结果(通过计算由击球手的击中数、出局数、击球数及坏球数体现的击球总数来确定)等计算。
- ◆ 提供一个允许用户使用 COM 组件的对话框。可以使用这个用户接口来代替强迫客户创建它。记住,应该总是力争实现代码的可重用性,甚至是在最小的编程方面。
- ◆ 在客户应用程序中,创建用户接口,以使它不需要选项卡。您可以拥有一个根据要执行的选定计算隐藏或显示字段的对话框。

下一章的实例将向您介绍如何通过 COM 组件来展示用户接口。

第 14 章 Visual J++ 应用程序的 高级实例

知识要点:

- ◆ 由实例背景提出需求分析
- ◆ 前期规划应考虑的问题
- ◆ 如何制定开发计划
- ◆ 软件开发过程
- ◆ COM 组件的高级应用

在本章中,我们将创建一个简单的示例,它合并了窗体设计、自定义控件创建、数据库访问,以及 COM 组件创建和实现等功能,是一个大型订单类型的应用程序。

因为该应用程序相对较大,并且覆盖了广泛的功能,所以本章涉及的示例与其他章节中的示例稍有不同。代替逐步进行每项操作,我们将从常见的开发情况开始,讨论开发应用程序的计划需求,并研究代码中有意义的功能。查看完代码,我们将熟悉应用程序,并弄清楚它具有的多项功能。

本章的目标有两个。首先我们将学习开发应用程序的正确方法,我们提供的方法与很多专业程序员用来创建实际应用程序的方法相似。其次本章升华了我们在前面章节中进行的工作,并涉及到很多更大型的应用程序中的功能。

下面我们先看一看实例背景。



光盘 参阅本书配套光盘中的【创建数据库应用程序】可交互学习与本章本关的知识。

14.1 实例背景

为了从本章获得更多的知识,您可以将自己置身于下面的角色中。您是一家中型图书商店 My Virtual Book Company 的顾问。新的 CEO 希望将公司的计算机和机构内部的应用程序进行更新。多年来,公司都是结合使用 DOS 和 Microsoft Windows 3.x 应用程序来处理库存、运输管理、销售合同和订单等。公司使用了一组小型的 Microsoft Access 数据库,每个应用程序拥有一个。因为每个应用程序都需要类似的数据,所以最近很多数据库被合并成一个 Access 数据库。在不久的将来,My Virtual Book Company 希望将数据库扩展到 Microsoft SQL Server。

新的 CEO 雇用您来开发第一批取代 DOS 和 16 位 Windows 应用程序的新的机构内部使用的应用程序系列。不幸的是,转移到 32 位应用程序将是一个很慢的过程,因为公司不可能停止所有的业务来过渡到新的软件。

开始,CEO 要求您为电话销售代表开发订单应用程序。该应用程序应该比较简单,因为用户的计算机知识可能很有限。CEO 要求该应用程序利用公司新近合并的 Microsoft Access 数据库中的现有数据结构。数据库将被放到 Microsoft SQL Server 中,并且公司希望尽量不要更改现有的数据库结构。

因为有些人要使用这个订单应用程序,所以 CEO 希望最大限度地防止对公司现有程序潜在的攻击和破坏。因此,该应用程序将只具有添加新订单和查看现有订单的能力。目前,当需要删除或更新订单时,它必须提交给经理。这种保障防止了对现有客户记录或可能已经发货的订单进行潜在的错误更改。编写完该应用程序后,如果公司对您的设计和实现感到满意,那么您将会被要求为自动更改订单记录的订单系统编写管理终端。

为了简化销售人员对应用程序的使用,CEO 希望您在用户界面设计中合并一些功能,让界面变得更简单明了。并且 CEO 希望应用程序能够提供一些统一的方法来执行常见的任务,如在数据库中搜索客户和图书等。

因为销售人员需要比较现有的和过去的订单信息来获得客户的订单变化,所以他们还需要创建过去客户订单的报表。目前公司正在评估所有要使用的翻新应用程序的报表工具,所以您现在必须改进它。如果需要您还必须创建一些自定义组件。公司希望尽量不要使用各种自定义控件,除非在用于应用程序之前得到彻底地测试。因为该产品是第一个要转换的,所以您需要自定义控件。

在机构内部应用程序被合并到日常业务中后,CEO 可能还希望能够通过公司的 Web 站点来购买图书。在您计划使用哪些编程语言和工具之前,考虑这种功能是很重要的。为机构内部使用而开发的大多数应用程序都可以很容易地传

输到 Web 中。对于您正打算编写的订单应用程序更是如此。

简单地说,该应用程序将利用公司当前可用的资源;它应该很容易使用,功能还要足够强大,以处理图书订单所需的多种类型的信息。图 14.1 显示了我们开发的满足这些要求的应用程序在运行时的外观。我们将这个应用程序称为 Book Order Manager,或简称为 Order Manager。电话销售人员在工作期间使用 Order Manager 就足够了。



图 14.1 运行时的 Book Order Manager 应用程序

为了使您的开发工作尽快走上正轨,需要计划一下应用程序的设计方法。这种前期规划很重要,因为它可以解答诸如“该版本的应用程序的功能有哪些?”、“为编写这个应用程序我们要利用哪些工具?”以及更重要的“采用哪种编程语言?”等问题。当您顺利确定采用的方法后,您就可以准备开发应用程序了。下面我们来详细地介绍一下前期规划。

14.2 前期规划

前期规划第一步也是最重要的一步是定义要在程序中提供的功能列表。该列表有两个方面是很重要的。首先,它使您了解应用程序的功能有哪些。您应

该能够像在 Microsoft 常说的“牢牢把握应用程序”。这意味着您应该非常清楚您的应用程序是什么。还要熟悉应用程序的特殊要求及其目的和功能。您的应用程序是一个实际存在的对象,而不是蒸气或没有形状和物质的东西。如果公司其他部门的人员需要了解程序的功能,那么这个信息是非常有用的。如果这些人员需要使用该应用程序,那么您或许希望在计划这个功能列表时考虑一下他们的建议。清楚地确定应用程序的功能还可以使您更有效地确定开发的重要过程及发布时间(如果该应用程序很大,需要大量时间)。

建立完功能列表后,宣布一下,使每个人都了解您要开发的是什么。然后像堡垒那样将它保护起来。我们在开发应用程序时遇到的最大的一个敌人是“功能蔓延”。如果您没有宣布功能并接受合理的反馈,那么通常需要在应用程序设计过程中添加新的功能或进行大的更改。因为随便进行更改,将在开发应用程序时花费更长的时间。我们通常会保持一个后备列表,通过“在下一版中考虑”这种方法,我们可以向人们表示正在考虑他们的意见。注意,使用户掌握应用程序比使它达到完美更重要。正是因为这种情况,我们减少了一些功能。当在本章稍后涉及到应用程序的高级设计及其用法时,我们将详细介绍它们。

开发、宣布并修改应用程序的功能列表后,您需要准备一下用来编写程序的工具。考虑一下功能列表,看一看可用的工具,然后确定是否需要更多的工具来实现列表中的功能。例如,如果需要提供特定类型显示的自定义控件,那么您需要查看是否可以使用现有的工具,否则,需要购买一些工具来满足需要。

对工具的需求随公司预算的不同而有所不同。如果您发现可用的工具不够,并且预算有限,那么可以考虑删减一些功能并将它们按优先级进行排序。这里使用 Microsoft Visual Studio 6.0 和 Microsoft Office 2000。

现在我们已经定义了功能列表,并根据可用的工具对它进行了评价,下面我们需要确定使用哪种编程语言。虽然这是一本关于 Visual J++ 和 WFC 的书,但我们想强调的是,在计划应用程序时,需要确定将要使用的编程语言。您需要根据功能列表进行比较来确定使用哪种编程语言更合理。哪一种语言提供最灵活最轻松的开发环境呢?您需要多少代码基础的控件呢?

像 Visual Basic 这样的工具也许可以满足应用程序的需要,如果应用程序将运行在企业网上,那么甚至简单的 HTML 就可以做到。还需要考虑到工具中自定义组件的可用性的因素,特别是您已有的自定义组件。我们这里的应用程序非常适合在 Visual J++ 和 WFC 中开发。由于预算的限制,它并不是很灵活。我们还必须为其他内部应用程序提供一些可重用的组件,以便整个公司存在一个通用的用户界面。

最后,CEO 希望能够将这个应用程序传送到公司的 Web 站点上,以便进行联机订购。具有 WFC 创建小型用户界面、易于设计的自定义控件、强大的 COM

组件,以及方便地访问任何类型的数据库的能力,所以 Visual J++ 是一个完美的工具。为了适应将来能够部署到 Internet 上的需要,我们还使用了 WFC 的 DHTML 类。

另外,如果您熟悉的编程语言不能满足应用程序的需要,那么应该重新评价一下您的功能列表,并进行必要的调整。但是现在您应该比较倾向于使用 Visual J++, 所以不必担心会减少功能。Visual J++ 几乎可以完成您需要的所有任务(当然,实际开发时可能会有出入)。

14.3 开发计划

现在我们已经完成了前期规划,下面可以集中精力确定开发方法。在设计阶段,首先需要确定开发的功能,然后确定如何将它们放到大型的应用程序设计中。如果开发应用程序的时间比较宽松,那么还应该考虑设计出用户界面的原型并得到反馈意见。Microsoft 经常使用用户的反馈意见来评价它的界面,以确保大多数人能够适应它的界面和功能。原型不需要与最终的设计完全相同,但是它们应该是相近的。从用户那里获得原型的反馈意见后,就可以全面地开始设计界面了。

我们开发的是一个 MDI 应用程序,它将显示包含所有订单处理相关信息的子窗口。这是应用程序提供的用户界面的高级视图。为了提供一致的客户端和图书搜索用户界面,我们将开发一些 COM 组件。还将使用一个限制条目数的自定义 WFC 控件。定义完该信息后,就可以更有效地计划首先设计的应用程序部分。通常的建议是首先设计所有的自定义控件,尤其当它们要被集成到子窗体中时。

下面我们来创建应用程序子窗体的用户界面及所需的支持对话框。该界面是应用程序中最重要的元素,因为它接受来自用户的数据。设计完子窗体后,创建将要与应用程序进行交互的 COM 组件。设计完子窗体的用户界面并明确应用程序将要进行的操作后,您可以将 COM 组件模块化以适应您的应用程序的需要,然后编写子窗体与 COM 组件进行交互的代码。

接下来添加 MDI 窗体及应用程序需要的所有其他支持对话框。MDI 窗体是子窗体与应用程序中其他对话框的连接点。最后将所有非用户的界面功能(如报表和数据验证例程等)编写到应用程序中。这些功能不会影响应用程序的核心架构,所以应该在最后添加它们。

完成开发顺序列表后,可以安排设计程序每个元素的阶段。在开发应用程序时,安排设计元素的时间(即使是宽松的)通常是一个好方法,即使您不希望是

开发过程中显示应用程序的安排或进度。在安排设计阶段和进行开发时,通常应该向人们展示一个结构图。您无法知道用户会在什么时间来了解工作的进展情况,并要求提供演示版。通常安排一下设计阶段,以便您可以在必要时提供演示。

现在我们准备使用为应用程序设计的功能列表和开发进程顺序来创建应用程序。在下一节中,将提供图书订单应用程序代码的重要部分。这个信息的顺序与刚才讲到的开发顺序是相同的。在讨论过程中,我们还将强调在程序每个部分实现的功能。通过这个概述,您应该理解用来开发应用程序的功能列表。

14.4 开发过程

正如在本章开始所讲到的,为适应本章的情况开发的示例应用程序的需求非常大。我们将不再逐步介绍示例的每一个开发过程,而是只强调一下开发过程的重点有一些有意义的代码。除此之外,在很多情况下,重新逐步创建应用程序将重复大量已经介绍过的内容,并且在各自的章节都有过详细的介绍。我将仅仅展示一些有意义的代码,并对它们进行研究。

完成开发过程的概述后,回顾一下用户将要对应用程序进行哪些操作。建议将示例从本书附带的光盘复制到系统中,以便您可以跟随我们的讨论顺序。除了工程文件,还需要在指定的地方放置一些文件。下一节将介绍如何处理这些事务。

14.4.1 准备

在可以开发应用程序之前,我们需要将一些文件放到\Windows\Application Data 目录中。回顾第 11 章“数据库编程”,Application Data 目录存储了所有应用程序指定的数据,并且它还是放置 Access 或索引顺序访问方法 (ISAM) 数据库及应用程序所需的常见文件的地方。在\Windows\Application Data 目录(或者在 Windows NT\WinNT\Profiles\<user>\Application Data 目录)下创建一个名为 DevWorkshop 的新目录。在这个目录下创建两个目录,一个名为 CustomerSearch,另一个名为 BookOrderManager。在 CustomerSearch 目录中,放置本书附带光盘 Samples\Chap14\Application Data\DevWorkshop\CustomerSearch 目录下的 FindFile.avi 文件。在进行搜索时,应用程序的 Customer Search Engine COM 组件将使用这个文件。该 Customer Search Engine COM 组件是第 13 章“COM 组件开发”中设计的 Customer Search Engine 的修改版。

将 BookOrders.mdb、Clouds.bmp、Logo.bmp 和 FormReuse.dll 文件从 Samples \ Chap14 \ Application Data \ DevWorkshop \ BookOrderManager 目录复制到您计算机上的 BookOrderManager 目录。BookOrders.mdb 是应用程序使用的数据库文件。该数据库中有一个包含公司所有客户的 Customer 表,一个列出所有订单的 Orders 表,以及一个列出每个订单购买的详细项目的 OrderDetails 表。

该数据库文件还包括列出所有用户可以购买的图书的 Books 表,以及列出所有允许使用该应用程序的雇员及其访问密码的 Employees 表。FormReuse.dll 文件包含在第13章设计的 FormReuse COM 组件中,该文件与 Logo.bmp 和 Clouds.bmp 文件一起用来显示应用程序的启动屏幕。



注意 如果您没有设计并打包第13章中的 FormReuse COM 组件,那么需要注册 FormReuse.dll 文件,以便应用程序可以使用它。使用 RegSvr32 应用程序来注册 COM 组件。在 Windows [运行] 对话框中键入 "RegSvr32 < PathToComponent >", 其中 < PathToComponent > 是 FormReuse.dll 文件的路径。

回顾我们设计的订单开发列表,我们将首先创建应用程序需要的所有自定义控件。这里只需要一个限制条目数的自定义控件。我们将使用这个控件来输入诸如邮政编码和密码等信息。回顾第8章“WFC 控件开发”,我们创建了 NumText 控件来解决这个问题。我们将在 Book Order Manager 应用程序中重新使用这个控件。如果在第8章您没有创建这个控件,那么强烈建议进行这项操作,以便理解通常在创建自定义 WFC 控件时涉及的过程。

因为 NumText 控件不应在包含它的项目之外打包,所以需要将项目的输出文件(.class 和 .tlb)一起打包到 zip 文件中,然后将这个 zip 文件移动到 Java 类的路径(通常为 \Windows \Java \Classes)中。这将使 Visual J++ 的 Customize Toolbox 对话框能够定位控件。在本书附带的光盘中提供了这样一个 zip 文件。它位于 \Chap14 \NumText 目录下,名为 NumText.zip。后面将会使用这个自定义 WFC 控件来开发其他 WFC 应用程序。

14.4.2 Order.java 对话框用户界面设计

现在辅助的文件已经各就各位,并根据需要进行了注册,下面我们来讨论项目开发及其重要的窗体和子窗体。如果您还没有做这些,可以在 Visual J++ 中打开该项目。我使用 Empty Project 模板创建了应用程序的项目,并将它命名为 BookOrderManager。应用程序的子窗体位于 Order.java 文件中。如果在 Forms Designer 中打开这个窗体,那么将会注意到对话框中包含3个选项卡:一个用于客

户信息,一个用于订单信息,另一个用于运输和支付信息。图 14.2~图 14.4 显示了 Order 对话框在开发期间的 3 个不同的选项卡。

The image shows a Java Swing window titled "Order" with three tabs: "Customer Information", "Order Information", and "Shipping & Payments". The "Customer Information" tab is selected. It contains several text input fields: "First Name", "Last Name", "Company", "Address 1", "Address 2", "City", "State" (a dropdown menu), "Zip Code", "Phone", and "Email Address". At the bottom, there is a large text area labeled "Notes".

图 14.2 Order 对话框在开发期间的 Customer Information 选项卡

The image shows the same "Order" window, but with the "Order Information" tab selected. It features a table with four columns: "Book Title", "Quantity", "Unit Price", and "Total Item Price". Below the table is an "Add" button. At the bottom of the window, there is a "Total Price of Order" label and a text field displaying "0.00".

图 14.3 Order 对话框在开发期间的 Order Information 选项卡



图 14.4 Order 对话框在开发期间的 Shipping & Payments 选项卡

Customer Information 选项卡几乎都是 Edit 控件,只有 Zip Code 字段使用 NumText 控件来限制条目只能有 5 位。Order Information 选项卡包含一个 ListView 控件,它将存储项目(每一个都是由一本特定的书组成)、订购的书的数目、单价,以及所订图书的总金额等。Add 按钮通过显示在本章稍后讨论的 Book Search Engine COM 组件来将项目添加到列表中。选项卡底部的 Label 控件动态地保持添加到列表中的所有项目的总金额。稍后我们将会看到这个变化的总金额如何得到更新。该对话框中最后一个选项卡是 Shipping & Payments 选项卡,像第一个选项卡那样,它包含一些用于输入数据的 Edit 控件。Shipping & Payments 选项卡将 NumText 控件用于 Shipping Zip Code 和 Credit Card Number 条目字段。

该对话框还包含 DataSource 和 ImageList 控件。DataSource 控件将与数据库的 Customers 表和 Customer Search Engine COM 组件进行交互。ImageList 将图标添加到显示所有订购的项目的 ListView 控件中:Single Book 图标用于订购一本图书的订单,Stack Of Books 图标用于包含多本图书的订单。这些图标可以很好地提示用户订单包含的内容,尤其是对大的订单。

另外,Order 对话框是一个子窗体,它将通过应用程序的 MDI 窗体来显示(稍后将介绍)。现在我们已经完成了界面子窗体的设计,下面来看一看应用程序中的子窗体和其他对话框将要与其进行交互的 COM 组件的设计过程。

14.4.3 COM 组件设计

因为我们在第 13 章已经正确地设计了 Customer Search Engine COM 组件,所以我只需对该项目做一些小的更改,以创建 Customer Search Engine 和 Book Search COM 组件。在 Customer Search Engine 中,更改数据源以使它指向新的数据库 BookOrders.mdb,并将搜索字段更改为数据库 Customers 表的字段。为了完成搜索字段中的更改,要更改代码来提供数据库中正确的字段,还要将搜索值更改为 Edit 控件的标签,以便它显示适当的提示。最后更改 DataGrid 控件中的列及其绑定字段。

为了将这个 Customer Search Engine 与第 13 章中的区别开来,还要将 COM 组件的项目名称更改为 CustomerSearchEngine2。通过在 Project Properties 对话框的 COM Classes 选项卡中取消并重新将它指定为 COM 类,并使用 Visual J++ 重新创建类的 @com.register 标记。这就是我们所要做的一切。因为代码比较容易更改,并且 COM 组件界面也不需要实际的更改,所以该 COM 组件很容易传输。

设计完 Customer Search Engine COM 组件,创建 Book Search Engine COM 组件就很容易了。只需将项目复制到新的项目目录中,并将它添加到 BookOrderManager 解决方案中。然后重新命名项目中所有文件,并对项目进行与 CustomerSearchEngine2 项目相同的更改。唯一的区别是,BookSearchEngine 需要更少的搜索字段,并且因为它它是通过 CustomerSearchEngine2 项目为基础而设计的,所以它的数据库位置不需要重新定义。它只需将它的查询更改为数据库中的 Book 表来代替 Customer 表。

将 CustomerSearchEngine2 和 BookSearchEngine COM 项目都添加到 BookOrderManager 解决方案中,然后可以同时编译它们和 BookOrderManager 项目。现在这两个 COM 组件的设计可以被公司任何未来的应用程序重复使用,并且它为应用程序提供了一个标准的用户界面。

14.4.4 代码的重要部分: Order.java 窗体代码

现在已经设计完应用程序的 COM 组件,下面来看一看为 Order.java 子窗体添加的一些较重要的代码。首先来看一看 showNewCustomerSearchForm 方法的代码:



```
/* * A new customer is being entered
 */
public void showNewCustomerForm( )
```



```

/* New customer, so set up data source
 * connection string by hand */
dsMain.setConnectionString(
    "Provider=Microsoft.Jet.OLEDB.3.51;"
    + "Persist Security Info=False;"
    + "Data Source="
    + SystemInformation.getSpecialFolderPath(
        SpecialFolder.APPLICATIONDATA)
    + "\\Dev\\Workshop\\BookOrderManager\\BookOrders.mdb");
// Set the mode of the order
miOrderMode = NEWCUSTOMERNEWORDER; // Set the caption of the
window
this.setText("New Customer: New Order * $");
// Show the child window
this.show();

```

MDI 窗体调用 `showNewCustomerForm` 同时创建新的订单和客户。因为这是一个新客户,所以没有使用 Customer Search Engine COM 组件,而是将该方法的代码重新设置 `DataSource` 控件的 `connectionString` 属性。代码将订单的模式设置为 `NEWCUSTOMERNEWORDER`。这将使保存代码能够确定保存订单的方式。然后代码更改 Order 窗体的标题,以显示该窗体将要接受新客户的新订单,并以非模块化的方式显示该窗体。

该方法的相反过程是 `showExistingCustomerForm`。`showExistingCustomerForm` 的代码如下所示:



```

/* * An existing customer is being used to create the order
 * */
public void showExistingCustomerForm(IDataSource db)
{
    // Set the mode of the order
    miOrderMode = OLDCUSTOMERNEWORDER;
    /* Assign the recordset control
     * to the form's data source control */
    dsMain = (DataSource) db;
    // Load customer's information into controls
    LoadCustomerInformation();
    // Show the child window
    this.show();
}

```

该方法接受实现 `IDataSource` 界面的对象。这里,它是 Customer Search Engine COM 组件返回的 `DataSource` 对象。当搜索返回有效的客户时,MDI 窗体将调用

该方法。代码将 Order 窗体的模式设置为 OLDCUSTOMERNEWORDER。因为现在已经具有一个有效的客户,所以不必创建一个新的客户——只需做一些适当的更改。

然后代码将 Order 窗体的 DataSource 控件设置为 MDI 窗体传递方法的 DataSource 对象。然后代码调用 LoadCustomerInformation 方法,将所有的客户信息从数据库中加载到 Customer Information 选项卡的字段中。最后 showExistingCustomerForm 显示窗体。

showNewCustomerForm 和 showExistingCustomerForm 方法显示它们所属的非常规的子窗体。当您需要根据应用程序的情况初始化一个不同的子窗体时,可以使用这样的方法来接收 MDI 窗体的重要信息,并将其显示出来。通常不需要初始化并显示 MDI 窗体的子窗体。这些方法同样适用于对话框。

这时,我们可以显示 MDI 父窗体的 Order 窗体。下面来看一看添加订单信息的代码。以下是 Order Information 选项卡中 btnAdd Button 控件 click 事件处理程序的代码:



```
/* * Adds an order detail item to the order
 * /
private void AddItemToOrder(Object source, Rvent e)
{
    // Create an instance of the NewOrderDlg dialog box
    NewOrderDlg orderDlg = new NewOrderDlg( );
    // Show the dialog box to retrieve items
    orderDlg.ShowDialog( );
    // If user clicked OK button in dialog box, proceed
    if (orderDlg.DialogResult( ) == DialogResult.OK)
    {
        // Create a ListItem object to store the values
        ListItem newItem = new ListItem( );

        // Set the text of the ListView item from NewOrderDlg
        newItem.setText(orderDlg.getBookTitle( ));
        /* Display a single book icon for single orders
         * and multiple books for a multibook order */
        if (orderDlg.getQuantity( ) > 1)
            newItem.setImageIndex(1);
        else
            newItem.setImageIndex(0);
        //Specify sub item values
        newItem.setSubItem(0,
            String.valueOf(orderDlg.getQuantity( )));
        newItem.setSubItem(1,
```

```

        Value.formatCurrency(orderDlg.getUnitPrice( ));
        newItem.setSubItem(2,
            Value.formatCurrency(orderDlg.getTotalPrice( )));
        /* This column is hidden and is used
         * to store the BOOK ID for the database */
        newItem.setSubItem(3,
            String.valueOf(orderDlg.getBookID( )));
        //Add the new item to the list
        lsvOrderItems.addItem(newItem);
        //Update total price label
        UpdateOrderPrice( );
        //Enable the Remove button
        btnRemove.setEnabled(true);
    }
}

```

当用户单击 Order Information 选项卡中的 Add 按钮来添加订单项目时, WFC 将调用 AddItemToOrder 事件处理程序。方法获得订单项目信息并更新选项卡中的 ListView 控件。代码将创建 NewOrderDlg 的实例并将其模式显示出来。图 14.5 显示了运行时的对话框。



图 14.5 运行时的 Create New Order 对话框

该对话框(将在本章最后的部分讨论)获得客户想要添加到订单中的图书信息。如果用户单击对话框中的 Add to List 按钮,那么代码将继续创建 ListView 控件中表示图书订单信息的 ListItem 对象,然后通过在新 OrderDlg 对话框中定义的上一级方法来设置 ListItem 及其子项目的信息。代码用数据完成项目后,它将把项目添加到 ListView 控件 lsvOrderItems。现在 ListView 控件中已经具有项目,代码将调用 UpdateOrderPrice 方法来更新显示添加到订单中的所有项目的总金额的 Label 控件。下面是 UpdateOrderPrice 方法:



```

/* * Updates the Order Price Label and Amount Due labels
 */
private void UpdateOrderPrice( )
{
    // Get all items in the ListView control
    ListItem[] tempItems = lsvOrderItems.getItems( );
    // Temporary total price
    double tempTotal = 0.00;
    // Loop through all items and calculate total price
    for (int i = 0; i < tempItems.length; i++)
    {
        /* Remove the dollar sign
         * so that we can calculate this as a double value */
        String tempString = tempItems[i]
                               .getSubItem(2)
                               .substring(1,
                                           tempItems[i]
                                               .getSubItem(2).length( ));

        // Get the float value for the item in the list
        double tempVal = Double.valueOf(tempString)
                               .doubleValue( );
        tempTotal = tempTotal + tempVal;
    }

    //Format the total price as currency
    lblTotalPrice.setText(Value.formatCurrency(tempTotal));

    //Determine if shipping charges have been added
    if (txtFreight.getText( ).length( ) != 0)
    {
        // Use this value to store the total amount due
        double tempAmountDue = 0.00;
        // Calculate the total amount due
        tempAmountDue = tempTotal
            + Double.valueOf(txtFreight.getText( ))
                .doubleValue( );
        // Format this value as a currency
        lblAmountDue.setText(Value
            .formatCurrency(tempAmountDue));
    }
    else
    {
        // Return the currency format for the total
        lblAmountDue.setText(Value.formatCurrency(tempTotal));
    }
}

```

2000

```
// Load the ship information from the customer info
```

```
LoadShipInfoFromCustomer();
```

ActiveTabChanged 方法是 Tab 控件 SelectedIndexChanged 事件的处理程序。当 Tab 控件的活动选项卡改变时,将触发该事件处理程序。该方法的代码确定新的活动选项卡是否为 Shipping & Payments 选项卡,以及窗体的 mbCustomer2Ship 标记变量是否被设置为 false。mbCustomer2Ship 变量确定用户是否已经得到使用订单运输字段的客户信息的提示。用户只被提示一次。如果两个条件都为真,那么代码将调用 LoadShipInfoFromCustomer 方法,它将简单地将 Customer Information 选项卡中相应的字段复制到 Shipping & Payments 选项卡中的运输信息字段。

输入完所有必要的信息后,可能会把订单保存到数据库中。以下是调用的 SaveOrder 方法:



```
/* * Starts the save operation for the order.
 * Returns true if order succeeds.
 */
public boolean SaveOrder()
{
    boolean returnVal = false;

    // Validate customer information
    if (ValidateCustomer())
    {
        // Validate order, ship, and payment information
        if (ValidateShipAndPay())
        {
            // Save customer information
            if (SaveCustomerInfo())
            {
                // Save order and order details to the database
                if (SaveOrderInfo())
                {
                    /* This flag notifies the
                     * closing event handler
                     * not to save again */
                    mbDirty = false;
                    returnVal = true;
                    this.dispose();
                }
            }
        }
    }
}
```

```
return returnVal;
```

```
|
```

通过 MDI 窗体来调用该方法。它是数据验证和订单信息保存方法的起始点。如果任何一个方法失败,那么 SaveOrder 将把一个错误值返回给 MDI 窗体,以表明出现问题或数据验证失败。如果所有的方法都返回真值,那么将关闭 Order 窗体,并将 true 值返回给 MDI 窗体。下面我们进一步来看一看数据验证和保存的方法。

虽然现在讨论的是数据验证的方法,但是您应该考虑在确保成功地将记录保存到数据库中之后再添加它们。通过将数据验证添加到应用程序中,您可以在创建订单的早期发现错误,这是数据验证程序的主要工作。下面我们来看一看 SaveOrder 方法中调用的每个方法的代码。以下是 ValidateCustomer 方法的代码:



```
/* *Validates data in the customers tab
*/
private boolean ValidateCustomer( )
|
    Control valCtrl;
    String tempString;

    // Check fields that are required for no value
    if (txtFirstName.getText( ).length( ) == 0)
    |
        tempString = "first name";
        valCtrl = txtFirstName;
    |
    else if (txtLastName.getText( ).length( ) == 0)
    |
        tempString = "last name";
        valCtrl = txtLastName;
    |
    else if (txtAddress1.getText( ).length( ) == 0)
    |
        tempString = "first address line";
        valCtrl = txtAddress1;
    |
    else if (txtCity.getText( ).length( ) == 0)
    |
        tempString = "city";
        valCtrl = txtCity;
    |
    else if (cmbState.getSelectedIndex( ) == -1)
```



```

    |
    |     tempString = "state";
    |     valCtrl = cmbState;
    |
    | else if (txtZipCode.getText().length() == 0)
    |
    |     tempString = "zip code";
    |     valCtrl = txtZipCode;
    |
    | else if (txtPhone.getText().length() == 0)
    |
    |     tempString = "phone number";
    |     valCtrl = txtPhone;
    |
    | else
    |     return true;
    |
    | /* Display a message to the users
    |  * notifying them of the changes needed */
    | MessageBox.show("You must specify a valid "
    |                 + tempString + ".",
    |                 MDIMain.msAppName
    |                 MessageBoxButtons.OK);
    | /* Ensure that the first tab is displayed
    |  * so that the user can see the problem */
    | tabMain.setSelectedIndex(0);
    | //Set the active control to the one that is invalid
    | setActiveControl(valCtrl);
    | return false;
    |

```

该方法非常直观,它说明了在数据验证例程中加强代码的主要办法。方法的前面部分包含了 Control 对象和 String 对象的声明。每次字段检查都将使用 Control 对象来指定 Customer Information 选项卡中的无效控件。String 对象保存无效控件的显示字段。如果 Customer Information 选项卡中的指定控件验证失败(例如它是空的),那么代码将存储在错误消息框中显示的字符串,还将存储无效控件的引用。

然后代码显示一个消息框,警告用户存在无效的值,选定 Customer Information 选项卡,以便用户可以看到出现的问题,然后将选项卡中的活动控件设置为存在无效数据的控件。如果出现数据验证错误,那么 ValidateCustomer 将返回 false 值。Customer Information 选项卡中只选中重要的或必需的字段,所以诸如 Notes 和 Email 等字段就没有什么价值。

第二个也是最后一个验证方法验证 Order 窗体的其他选项卡: Shipping &

Payments 和 Order Information。以下是 ValidateShipAndPay 方法的代码：



```

/* * Validates information on the Shipping & Payments tab
 * and the Order Information tab
 */
private boolean ValidateShipAndPay( )
{
    /* Determine whether the CreditCard fields
     * are available to validate */
    if (txtExpDate.getEnabled( ) != false)
    {
        /* Determine whether
         * the Expiration Date control is empty */
        if (txtExpDate.getText( ).length( ) == 0)
        {
            MessageBox.show("You must specify a valid credit "
                            + "card expiration date for the "
                            + "shipping information.",
                            MDIMain.msAppName,
                            MessageBox.ICONERROR);

            /* Ensure that the appropriate tab is displayed
             * so that the user can see the problem */
            tabMain.setSelectedIndex(2);
            setActiveControl(txtExpDate);
            return false;
        }

        // Validate the CreditCard expiration date field
        try
        {
            /* Create a time object with the date entered in
             * the control. If it fails, try/catch will handle
             * by displaying a message box. */

            Time timeExp = new Time(txtExpDate.getText( ));
            /* The date is valid.
             * Is it earlier than today's date?
             * Then it is expired. Notify user. */
            if (timeExp.compareTo(new Time( ).getDate( )) == -1)
            {
                MessageBox.show("This credit card "
                                + "has already expired!",
                                MDIMain.msAppName,
                                MessageBox.ICONERROR);

                /* Ensure that the appropriate tab is displayed
                 * so that the user can see the problem */
            }
        }
    }
}

```

```

        tabMain.setSelectedIndex(2);
        setActiveControl(txtExpDate);
        txtExpDate.selectAll( );
        return false;
    }

    catch (Exception err)
    {
        MessageBox.show(err.getMessage( ),
                        MDIMain.msAppName,
                        MessageBox.ICONEXCLAMATION);
        /* Ensure that the appropriate tab is displayed
         * so that the user can see the problem */
        tabMain.setSelectedIndex(2);
        setActiveControl(txtExpDate);
        txtExpDate.selectAll( );
        return false;
    }

// Validate the Shipping Date field
try
{
    /* Determine whether
     * the Expiration Date control is empty */
    if (txtShipDate.getText( ).length( ) == 0)
    {
        MessageBox.show("You must specify a valid"
                        + "shipping date for the"
                        + "shipping information.",
                        MDIMain.msAppName,
                        MessageBox.ICONERROR);
        /* Ensure that the appropriate tab is displayed
         * so that the user can see the problem */
        tabMain.setSelectedIndex(2);
        /* Select the Shipping Date control
         * as the active control */
        setActiveControl(txtShipDate);
        //Select all text in the control
        txtExpDate.selectAll( );
        return false;
    }

    /* Create a time object with the date entered
     * in the control. If it fails, try/catch
     * will handle by displaying a message box. */

```

```

        Time timeExp = new Time(txtShipDate.getText());
    }
    catch (Exception err)
    {
        MessageBox.show(err.getMessage(),
                        MDIMain.msAppName,
                        MessageBox.ICONEXCLAMATION);
        /* Ensure that the appropriate tab is displayed
         * so that the user can see the problem */
        tabMain.setSelectedIndex(2);
        setActiveControl(txtExpDate);
        txtExpDate.selectAll();
        return false;
    }

    //Validate that there are items ordered
    if (lsvOrderItems.getItemCount() == 0)
    {
        MessageBox.show("You must have items "
                        + "in order to complete the order!",
                        MDIMain.msAppName,
                        MessageBox.ICONEXCLAMATION);
        /* Ensure that the appropriate tab is displayed
         * so that the user can see the problem */
        tabMain.setSelectedIndex(1);
        setActiveControl(lsvOrderItems);
        return false;
    }

    // Validate the shipping fields...

    //Start with the freight field to check for a valid entry
    try
    {
        double tempVal = Value.toDouble(
                                txtFreight
                                .getText()
                                .substring(1, txtFreight
                                .getText()
                                .length()));
    }
    catch (Exception numErr)
    {
        MessageBox.show("You have specified an invalid format"
                        + "for the Freight of the order. "
                        + "Please re-enter the freight.",
                        MDIMain.msAppName,

```

```
        MessageBox.ICONEXCLAMATION);
    /* Ensure that the appropriate tab is displayed
     * so that the user can see the problem */
    tabMain.setSelectedIndex(2);
    setActiveControl(txtFreight);
}

Control valCtrl;
String tempString;

// Check fields that are required for no value
if (txtSFirstName.getText().length() == 0)
{
    tempString = "first name";
    valCtrl = txtFirstName;
}
else if (txtSLastName.getText().length() == 0)
{
    tempString = "last name";
    valCtrl = txtLastName;
}
else if (txtSAddress1.getText().length() == 0)
{
    tempString = "first address line";
    valCtrl = txtAddress1;
}
else if (txtSCity.getText().length() == 0)
{
    tempString = "city";
    valCtrl = txtCity;
}
else if (cmbSState.getSelectedIndex() == -1)
{
    tempString = "state";
    valCtrl = cmbState;
}
else if (ntxtSZipCode.getText().length() == 0)
{
    tempString = "zip code";
    valCtrl = ntxtZipCode;
}
else if (txtSPhone.getText().length() == 0)
{
    tempString = "phone number";
    valCtrl = txtPhone;
}
```

```

else if (ntxtCreditCard.getEnabled( )
        && ntxtCreditCard.getText( ).length( ) == 0)
{
    tempString = "credit card number";
    valCtrl = ntxtCreditCard;
}
else if (cmbPaymentType.getSelectedIndex( ) == -1)
{
    tempString = "payment type";
    valCtrl = cmbPaymentType;
}
else if (cmbShipMethod.getSelectedIndex( ) == -1)
{
    tempString = "shipping method";
    valCtrl = cmbShipMethod;
}
else
    return true;

/* Display a message to users
 * notifying them of the changes needed */
MessageBox.show("You must specify a valid"
                + tempString
                + " for the shipping information.",
                MDIMain.msAppName,
                MessageBox.ICONERROR);

/* Ensure that the appropriate tab is displayed
 * so that the user can see the problem */
tabMain.setSelectedIndex(2);
// Set the active control to the one that is invalid
setActiveControl(valCtrl);
/* Set focus to the correct tab so the user can see
 * where they need to make changes */
tabMain.setSelectedIndex(2);
return false;
}

```

ValidateShipAndPay 方法相当大,并且可以被分成 3 部分。第 1 部分处理验证信用卡的失效日期(如果从 cmbPaymentTypes ComboBox 控件中选择了信用卡)和运输日期字段。这部分代码将验证日期是否在于各自的字段中,并且验证输入到控件中的值是否可以被转换成有效值。对于日期验证,代码将输入的日期分配给 Time 控件。如果因为存在无效值而出现例外,那么方法将通知用户并返回 false 值。在信用卡失效日期字段中,代码还将通过比较输入的日期和当前日期来查看该卡是否过期。

ValidateShipAndPay 的下一部分代码检查用户是否将项目添加到订单中。如果订单中没有项目,那么它是一个无效的订单。代码通过获得 Order Information 选项卡 ListView 控件中的项目数来查看订单包含的项目。如果项目数为 0,那么代码将警告用户,将选项卡移到 Order Information 选项卡,然后返回 false 值。

代码执行的最后部分检查是否与 ValidateCustomer 方法的执行过程基本一致。

现在我们来查看 SaveCustomerInfo 方法,它将保存 Customer Information 选项卡中的客户数据。该方法的代码如下所示:



```

/* * Saves or updates customer information based on document mode
 */
private boolean SaveCustomerInfo( )
{
    try
    {
        /* If this customer is new, we need to create
         * a valid recordset to work with */
        if (miOrderMode == NEWCUSTOMERNEWORDER)
            dsMain.setCommandText("Select * FROM Customers");

        // Reference the data source's control
        Recordset tempRs = dsMain.getRecordset( );

        /* Determine whether the recordset
         * is pointing at the current customer's record */
        if (tempRs.getField("CustomerID").getInt( )
            != miCustomerID && miCustomerID != 0)
        {
            // Move to the current customer's record
            tempRs.find("CustomerID = "
                + String.valueOf(miCustomerID));
        }

        // If this customer is new, start the add new process
        if (miOrderMode == NEWCUSTOMERNEWORDER)
            tempRs.addNew( );
        /* STORE INFORMATION IN DATABASE */

        tempRs.getField("FirstName")
            .setString(txtFirstName.getText( ));
        tempRs.getField("LastName")
            .setString(txtLastName.getText( ));
        tempRs.getField("CompanyName")
            .setString(txtCompany.getText( ));
        tempRs.getField("Address")
            .setString(txtAddress1.getText( ));
    }
}

```



```

tempRs.getField("Address2")
    .setString(txtAddress2.getText());
tempRs.getField("City")
    .setString(txtCity.getText());
tempRs.getField("State")
    .setString((String) cmbState.getSelectedItem());
tempRs.getField("ZipCode")
    .setString(txtZipCode.getText());
tempRs.getField("Phone")
    .setString(txtPhone.getText());
tempRs.getField("Email")
    .setString(txtEmail.getText());
tempRs.getField("Notes")
    .setString(txtNotes.getText());
//Update the recordset into the database
tempRs.update();
/* Move to this record and
 * store customer ID if new customer */
if (miOrderMode == NEWCUSTOMERNEWORDER)
{
    tempRs.moveToLast();
    miCustomerID = tempRs.getField("CustomerID")
        .getInt();
}
}
catch (Exception err)
{
    MessageBox.show(err.getMessage(),
        MDIMain.msAppName,
        MessageBox.ICONERROR);
    return false;
}
return true;
}

```

SaveCustomer 方法的代码非常直观,我们先来浏览一下。代码从查看订单是新客户的订单还是现有客户的订单开始。如果是新的客户,那么将使用基本记录集来初始化 Order 窗体的 DataSource 控件。这将使代码能够把新的客户记录添加到数据库中。如果是现有的客户,那么该代码就不是必要的,因为它已经存在要使用的有效记录集。

接下来代码将创建 DataSource 控件记录集的引用,以便它更易于访问。然后如果该订单不是新客户,那么代码将试着使用 Order 窗体的 miCustomerID 成员变量(存储当前客户的 ID 值)将记录集移动到当前订单的客户记录中。移动到

数据库适当的客户记录中后,就可以更新客户的字段值。

因为该客户也许是新客户,所以代码将进行检查,然后调用 `addNew` 方法来通知数据库:我们正在创建一个新的客户记录。代码将 `Order` 对话框中的不同字段分配给数据库 `Customer` 表的字段,并调用 `update` 方法来提交对数据库所做的更改。代码最后查看这个客户是否为新客户,然后移动到新创建的客户记录,并将该记录的 `CustomerID` 值存储到 `Order` 对话框的私有成员变量 `miCustomerID` 中。如果在执行该方法代码的过程中出现意外情况,那么代码将返回 `false` 值并警告用户。

下面讨论 `Order` 窗体的最后一个方法,也可能是整个应用程序中最大的方法, `SaveOrderInfo`。该方法保存 `Order Information` 和 `Shipping & Payments` 选项卡中的所有信息。以下是该方法的代码:



```

/* * Saves or updates order information
 */
private boolean SaveOrderInfo( )
{
    try
    {
        /* Create a temporary DataSource object to work with
         * instead of the Customers object */
        DataSource dsOrders = new DataSource( );
        //Set the connection string to point to our database
        dsOrders.setConnectionString(
            "Provider=Microsoft.Jet.OLEDB.3.51;"
            + "Persist Security Info=False;"
            + "Data Source="
            + SystemInformation.getSpecialFolderPath(
                SpecialFolder.APPLICATIONDATA)
            + "\\DevWorkshop\\BookOrderManager\\BookOrders.mdb");
        /* Create a recordset in the DataSource object
         * that points to the OrderDetails table */
        dsOrders.setCommandText("Select * FROM Orders");
        // Reference the data source's recordset
        Recordset tempRs = dsOrders.getRecordset( );

        // Start adding the record
        tempRs.addNew( );

        /* STORE INFORMATION IN DATABASE */

        // Get reference to MDI form to obtain employee ID
        MDIMain refMDI = (MDIMain) this.getParentForm( );
        tempRs.getField("CustomerID")
    }
}

```

```

        .setInt(miCustomerID);
tempRs.getField("EmployeeID")
        .setInt(refMDI.getEmployeeID());
tempRs.getField("OrderDate")
        .setString(new Time().formatShortDate());
tempRs.getField("ShipFirstName")
        .setString(txtSFirstName.getText());
tempRs.getField("ShipLastName")
        .setString(txtSLastName.getText());
tempRs.getField("ShipAddress")
        .setString(txtSAddress1.getText());
tempRs.getField("ShipAddress2")
        .setString(txtSAddress2.getText());
tempRs.getField("ShipCity")
        .setString(txtSCity.getText());
tempRs.getField("ShipState")
        .setString((String) cmbSState.getSelectedItem());
tempRs.getField("ShipZipCode")
        .setString(ntxtSZipCode.getText());
tempRs.getField("ShipPhone")
        .setString(txtPhone.getText());
tempRs.getField("ShipDate")
        .setString(txtShipDate.getText());
tempRs.getField("ShippingMethodID")
        .setInt(cmbShipMethod.getSelectedIndex());
tempRs.getField("FreightCharge")
        .setDouble(Value.toDouble(txtFreight.getText()));
tempRs.getField("PaymentType")
        .setInt(cmbPaymentType.getSelectedIndex());
/* Save these fields only if
 * the selected payment type is a credit card */
if (cmbPaymentType.getSelectedIndex() <= 2)
{
    tempRs.getField("CreditCard#")
        .setString(ntxtCreditCard.getText());
    tempRs.getField("ExpirationDate")
        .setString(txtExpDate.getText());
}
//Update the recordset into the database
tempRs.update();

// Get the order ID of the new order record
miOrderID = tempRs.getField("OrderID").getInt();

/* Create a temporary DataSource object
to work with instead of the Customers object */

```

```

DataSource dsDetails = new DataSource( );
// Set the connection string to point to our database
dsDetails.setConnectionString(
    "Provider=Microsoft.Jet.OLEDB.3.51;"
    + "Persist Security Info=False;"
    + "Data Source="
    + SystemInformation.getSpecialFolderPath(
        SpecialFolder.APPLICATIONDATA)
    + "\\DevWorkshop\\BookOrderManager\\BookOrders.mdb");
/* Create a recordset in the DataSource object
 * that points to the OrderDetails table */
dsDetails.setCommandText("Select * FROM OrderDetails");
// Reaquire recordset
Recordset tempRs2 = dsDetails.getRecordset( );
/* STORE INFORMATION IN DATABASE */

// Loop through all items in list view
for (int i = 0; i < lsvOrderItems.getItemCount( ); i++)
{
    // Start adding a new record
    tempRs2.addNew( );

    // Store values for selected fields of the ListView control
    tempRs2.getField("OrderID")
        .setInt(miOrderID);
    tempRs2.getField("BookID")
        .setInt(Value.toInt(lsvOrderItems.getItem(i)
            .getSubItem(3)));
    tempRs2.getField("Quantity")
        .setInt(Value.toInt(lsvOrderItems.getItem(i)
            .getSubItem(0)));
    String tempPrice = lsvOrderItems.getItem(i)
        .getSubItem(2);

    /* Remove dollar sign from the
     * total price field since it is a double */
    tempPrice = tempPrice.substring(1,
        tempPrice.length( ));

    tempRs2.getField("Price")
        .setDouble(Value.toDouble(tempPrice));

    //Update the database
    tempRs2.update( );
}

/* Dispose both data source objects
 * to clean up after our database connections */
dsOrders.dispose();

```

```

        dsDetails.dispose();
    }
    catch (Exception err)
    {
        MessageBox.show(err.getMessage(),
                        MDIMain.msAppName,
                        MessageBoxButtons.OK,
                        MessageBoxIcon.ERROR);
        return false;
    }
    return true;
}

```

SaveOrderInfo 方法的代码与 SaveCustomerInfo 方法的不同,它不需要知道窗体处于什么模式,因为它可以在两者的模式中将信息保存到新记录中。另一个不同是,代码使用运行时创建的独立 DataSource 对象来创建记录,并将它们分配到数据库各自的表中。如何在运行时创建 DataSource 对象来代替在设计时控件是很重要的。

代码的第一部分处理在 Order 中创建订单记录的任务。使用 Shipping & Payments 字段的信息正确地定义订单后,代码将检索并存储订单的 OrderID。然后该 OrderID 用于代码的下一部分,它将在 Order Information 选项卡 ListView 控件的所有项目中循环。对于 ListView 控件中的每一个项目,都会在数据库的 OrderDetails 表中使用刚创建的当前 OrderID 来创建记录。虽然该方法较大,但它很容易阅读和理解。

在结束 Order 窗体源代码的重要部分之前,来看一下如何使用 AddItemToOrder 事件处理程序来添加订单项目的 Create New Order 对话框(NewOrderDlg 对话框)。图 14.6 显示了该窗体在设计时的外观。

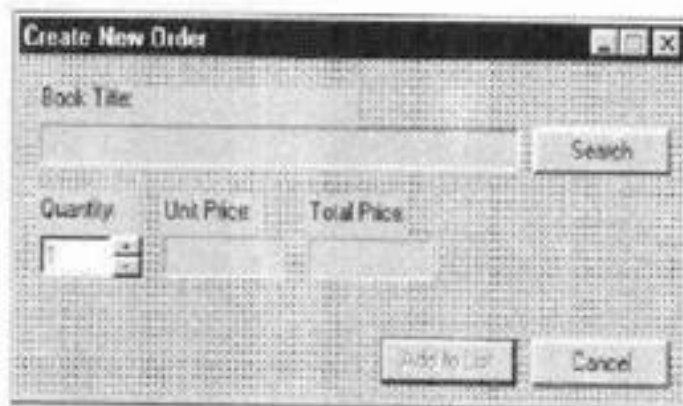


图 14.6 设计时的 Create New Order 对话框

该对话框提供了一个基本的用户界面,它显示了订购的图书名称、数量、单

价,以及订单总金额等。唯一可以通过该用户界面修改的字段是 txtQuantity Edit 控件。不能在其中键入数量,但是可以通过微调按钮来修改它。对于选定要订购图书的用户,只需单击 lblTitle Label 控件旁的 btnBrowse Button 按钮,btnBrowse 的 click 事件处理程序代码会创建 BookSearchEngine COM 组件的实例,并将它显示出来。

返回 Book Search Engine COM 组件后,btnBrowse 的事件处理程序将调用 LoadBookInfo 方法,它将所选图书的信息加载到对话框的控件中。LoadBookInfo 还将使用所选图书的数据来设置 3 个存储信息的私有成员变量,以便稍后 Order 窗体可以通过访问体方法来获得它。该对话框还包括根据当前选定的图书及用户指定的数量更新对话框中订单总额的代码。

New Order 对话框、Create New Order 对话框及 Order 窗体用于处理所有获得和保存数据的任务。接下来,我们介绍的用户界面元素是 MDI 窗体。

14.4.5 主要代码:MDIMain 窗体代码

该应用程序的 MDI 窗体 MDIMain.java 是 BookSearchManager 项目的核心。虽然 Order 窗体是应用程序用户界面最完整的部分,但是 MDIMain 维护应用程序各种元素间的关系。图 14.7 显示了设计时的 MDIMain.java。在此窗体中,我们使用了现成的 PersistentForm 模板的窗体(在 Samples \ Chap14 \ Persistent-FormTemplate 目录下),然后将 PersistentForm 模板的窗体进行了子类化。使用该模板,允许 MDIMain 重新调用它最后的位置及窗口状态,并在下一次显示该窗口时恢复它们。

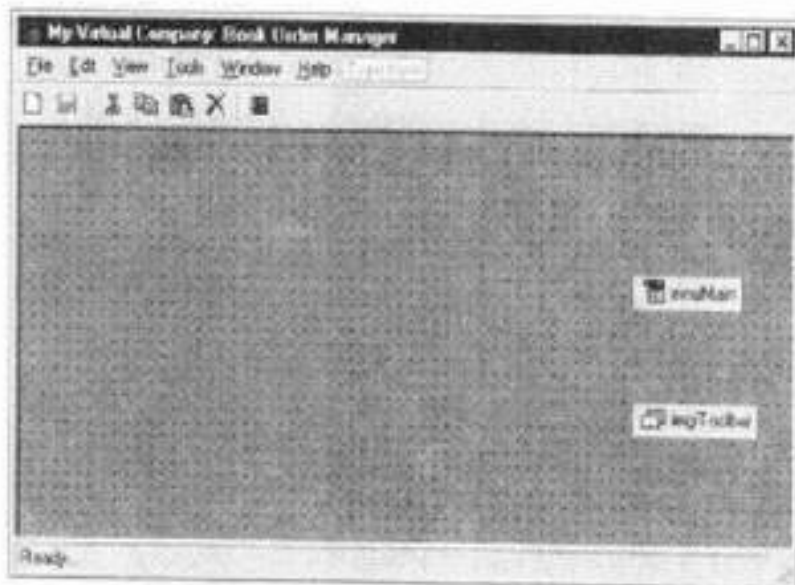


图 14.7 设计时的 MDIMain.java 窗体

正如您所看到的,MDIMain 窗体包含 MainMenu 控件、ToolBar 控件以及 StatusBar 控件。ImageList 控件也位于窗体中,它将存储 ToolBar 控件的图像。因为窗体的菜单相对简单,所以 ToolBar 控件包含了有限数量的按钮(经常使用的命令)。

ToolBar 控件中的 tbPrevOrd ToolBar 按钮是一个切换按钮,当显示 Previous Orders 工具窗口(稍后将在本章介绍这个窗体)时它呈现按下状态。该按钮为这个经常使用的窗口提供了快速访问的方法。StatusBar 控件只包含一个窗格,MDI 窗体用它来通知用户订单已经被保存,并且关闭了 Order 窗体。

下面我们来看一看它的代码。我们从窗体的构造函数开始:



```
public MDIMain(RegistryKey tempKey, String empName, int empNum)
{
    // Assign the Registry key that was passed as an argument
    persistKey = tempKey;
    // Call initForm to setup controls and other form properties
    initForm();
    // Load the stored window state (if any)
    LoadWindowState();

    //Store the employee name for use in the About box
    msUserName = empName;
    //Store the employee number
    miEmpNum = empNum;

    //Create instance of the splash screen
    FormReuse splash = new FormReuse();
    //Show the splash screen
    splash.showSplash(
        "My Virtual Company: Book Division",
        SystemInformation
            .getSpecialFolderPath(SpecialFolder.APPLICATIONDATA)
            + "\\DevWorkshop\\BookOrderManager\\Logo.bmp",
        SystemInformation
            .getSpecialFolderPath(SpecialFolder.APPLICATIONDATA)
            + "\\DevWorkshop\\BookOrderManager\\clouds.bmp",
        "Copyright 1998, My Virtual Company",
        "",
        msUserName,
        "Order Manager",
        "Version 1.0",
        5000);

    // Connect the onIdle event to an event handler
    Application.addOnIdle(new EventHandler(UpdateToolBars));
}
```


正如前面提到的,该窗体是通过 PersistentForm 模板获得的,所以构造函数包含作为参数的 RegistryKey 对象。我们还将添加两个提供当前登录到应用程序的雇员信息的参数。雇员使用 ApplicationLogin 窗体(稍后将在本章讨论)来进行登录。PersistentForm 模板提供了前几行代码,MDIMain 用它来支持恢复窗体的窗口状态设置。该代码将构造函数接收到的雇员名称和号码存储到它的参数列表中。然后窗体创建一个 FormReuse COM 组件的实例。我们在项目中为这个 COM 组件创建了 COM 封装程序。

COM 组件的实例使用该 COM 组件的 showSplash 方法显示启动屏幕。该 COM 组件从应用程序的数据目录中加载启动屏幕的位图。显示启动屏幕并返回后,代码将把 onIdle 事件与名为 UpdateToolBars 的事件处理程序联系起来。因为更新工具栏没有统一的位置,所以使用 onIdle 事件来处理启用和禁用工具栏的任务。当应用程序处理完所有的事件并有一个空的事件队列时,将发生 onIdle 事件。在 onIdle 事件中更新工具栏是一个常见的技巧,因为该事件会经常被触发。

在我们讨论 UpdateToolBars 方法之前,先来浏览一下它的代码:



```

/* * Handles updating the toolbars enabled state
 * and the Edit menu's state
 */
private void UpdateToolBars(Object source, Event e)
{
    // Determine if we don't have a valid MDI child
    if (getActiveMDIChild() == null)
    {
        // Disable the proper toolbars and menus
        if (tbSave.getEnabled() != false)
            tbSave.setEnabled(false);
        if (tbPrevOrd.getEnabled() != false)
            tbPrevOrd.setEnabled(false);
        if (tbCut.getEnabled() != false)
        {
            mnuCut.setEnabled(false);
            tbCut.setEnabled(false);
        }
        if (tbCopy.getEnabled() != false)
        {
            mnuCopy.setEnabled(false);
            tbCopy.setEnabled(false);
        }
        if (tbPaste.getEnabled() != false)
        {

```

```

        mnuPaste.setEnabled(false);
        tbPaste.setEnabled(false);
    }
    if (tbDelete.setEnabled() != false)
    {
        mnuDelete.setEnabled(false);
        tbDelete.setEnabled(false);
    }
    // Update the SelectAll menu item
    mnuSelectAll.setEnabled(false);
}
else
{
    // Check for the active control being an Edit control
    Order tempOrder = (Order) getActiveMDIChild();
    boolean editStatus = tempOrder.getActiveControl()
        instanceof Edit;
    /* Set the SelectAll menu based
     * on whether it is an Edit control */
    mnuSelectAll.setEnabled(editStatus);
    /* Handle the Paste menu differently
     * by checking for text on Clipboard */
    if (Clipboard.getDataObject()
        .getDataPresent(String.class)
        != true)
    {
        mnuPaste.setEnabled(false);
        if (tbPaste.setEnabled()) tbPaste.setEnabled(false);
    }
    else
    {
        mnuPaste.setEnabled(editStatus);
        if (tbPaste.setEnabled() != editStatus)
            tbPaste.setEnabled(editStatus);
    }
    /* Check to see whether we have an edit control
     * so we can check for selected text */
    if (editStatus == true)
    {
        /* Ensure that we have selected text for
         * Cut, Copy, and Delete */
        Edit tempEdit = (Edit) tempOrder.getActiveControl();
        if (tempEdit.getSelectedText().length() == 0)
            editStatus = false;
    }
}

```

```

/* Enable or disable the menu and controls
 * based on state and selected text */
if(tbCut.setEnabled() != editStatus)
{
    tbCut.setEnabled(editStatus);
    mnuCut.setEnabled(editStatus);
}
if(tbCopy.setEnabled() != editStatus)
{
    tbCopy.setEnabled(editStatus);
    mnuCopy.setEnabled(editStatus);
}
if(tbDelete.setEnabled() != editStatus)
{
    tbDelete.setEnabled(editStatus);
    mnuDelete.setEnabled(editStatus);
}

/* Enable the Save and Previous Orders menu items
 * since we have an order */
if (tbSave.setEnabled() != true)
    tbSave.setEnabled(true);
if (tbPrevOrd.setEnabled() != true)
    tbPrevOrd.setEnabled(true);
/* Since the Previous Orders window can be closed
by clicking the Close button, we need to update the
toolbar button. Check for the window being visible */
tbPrevOrd.setPushed(ordDlg.getVisible());

```

该事件处理程序的行为与其名称的含义相反。UpdateToolBars 方法在需要时禁用工具栏,它不处理 Edit 菜单项目的启用状态。该代码首先确定当前是否显示有子窗体开始。如果没有,那么代码将禁用除 tbNew 以外的所有其他工具栏按钮。在禁用按钮时,代码将检查并确保当前没有禁用按钮。这种检查防止了代码经常禁用 Toolbar 控件上的按钮时可能会导致的工具栏闪动。UpdateToolBars 还将检查在存在子窗体时启用按钮之前,按钮是否还没有被启用。同时代码更新 mnuCut、mnuCopy、mnuPaste、mnuSelectAll 和 mnuDelete 菜单项目的 enabled 状态。

如果存在订单窗体,那么代码将获得子窗体的实例,并查看窗体中当前的活动控件是否是 Edit 控件或其衍生物。如果是,那么代码将启用 mnuSelectAll 菜单项目。代码将查看【剪贴板】中是否有数据。如果【剪贴板】中有可以粘贴的数据,那么代码将启用 mnuPaste 菜单和 tbPaste ToolBar 按钮。

如果当前子窗体中的当前控件是 Edit 控件,那么代码将确定控件中是否有选定的文本。它通过创建控件的引用作为 Edit 控件来做到这一点。然后,代码将调用 Edit 类的 `getSelectedText` 方法。如果控件中没有选定的文本,那么代码将禁用 `mnuCut`、`mnuCopy` 和 `mnuDelete` 菜单项目及其相关的 ToolBar 按钮。否则,代码将启用它们。

如果出现 MDI 子窗体,那么 `UpdateToolBars` 还将启用 `tbSave` 和 `tbPrevOrd` 工具栏按钮。`PreviousOrders` 窗口有一个 Close 按钮,用户可以通过单击它来关闭窗口。为了防止实际关闭窗口,可以隐藏它。因为用户可能通过这种方法来关闭 `PreviousOrders` 窗口,所以我们需要在隐藏窗口时更新 `tbPrevOrd` 工具栏按钮。

下面将要介绍的代码是 `NewOrder` 事件处理程序的代码。当用户单击 `tbNew` 或 `mnuNewOrder` 时将调用该事件处理程序。该事件处理程序将启动新的订单过程。代码如下所示:



```

/* * Starts the new order process...
 */
private void NewOrder(Object source, Event e)
{
    // Prepare an instance of the child form
    Order dlgOrder = new Order();
    // Set the child's parent to be the MDI form
    dlgOrder.setMDIParent(this);
    // Create instance of the CustomerSearch COM component
    CustomerSearch csDlg = new CustomerSearch();
    // Show its search dialog box
    csDlg.showCustomerSearchDialog();
    // Determine whether this customer is new or existing
    if (csDlg.getSelectedCustomer() != null)
    {
        /* Give the child form the data source control
         * that the customer search dialog box returned */
        dlgOrder.showExistingCustomerForm(
            csDlg.getSelectedCustomer());
    }
    else // This customer is new
    {
        // Create a new instance of the form
        dlgOrder.showNewCustomerForm();
    }
}

```

该事件处理程序的代码创建 `Order` 窗体的实例,并调用它的 `setMDIParent` 方法来制作 `Order` 窗体子窗口的实例。接下来代码将通过创建使用将 COM 组件导

入到项目中的方法创建的 CustomerSearch COM 封装程序类的实例,来创建 CustomerSearchEngine2 COM 组件的实例。创建完 CustomerSearchEngine2 的实例后,代码将调用该 COM 组件的 showCustomerSearchDialog 方法来显示该组件的 Search For Customer 对话框。

如果用户在该对话框中选定有效的客户(通过将 COM 组件 getSelectedCustomer 方法与 null 进行比较来确定),那么 NewOrder 的代码将调用 Order 窗体的 showExistingCustomerForm 方法。该方法显示包含 NewOrder 传递给 DataSource 对象的客户信息的 Order 窗体。如果用户单击 CustomerSearchEngine2 中的 Cancel 按钮(这将使 CustomerSearchEngine2 返回空的 DataSource 对象),那么 NewOrder 的代码将调用 Order 窗体的 showNewCustomerForm 方法来启动空的 Order 窗体。

在结束讨论 MDIMain 的代码之前,还应该简要地介绍两个方法: ManagePreviousOrders 方法和 DisplayReports 方法。下面让我们从 ManagePreviousOrders 方法开始讲起。当销售人员查看客户当前订购情况时它通常很有用。BookOrderManager 应用程序有一个名为 Previous Orders 的工具窗口,它显示了当前活动 Order 窗体的这种信息。图 14.8 显示了运行时的 Previous Orders 工具窗口。

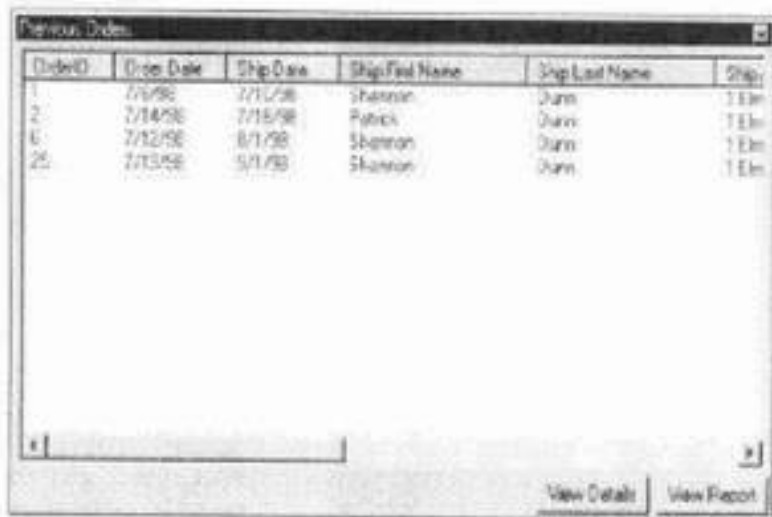


图 14.8 运行时的 Previous Orders 工具窗口

ManagePreviousOrdersWindow 的代码根据方法的调用情况来隐藏或显示工具窗口。该方法的代码还将根据它的调用情况来更新显示的内容。该方法的代码如下所示:



```

/* * Manages the display of the Previous Orders tool window
 */
public void ManagePreviousOrdersWindow(int mode)
{
    if (mode == POSHOWHIDE)
    {

```

```

/* Determine whether we have any active children
 * displayed and whether the menu item is selected */
if (getActiveMDIChild() != null
    && mnuPrevOrders.isChecked() == false)
{
    // Check the menu item
    mnuPrevOrders.setChecked(true);
    // Get a reference to the ActiveMDIChild
    Order tempOrder = (Order) getActiveMDIChild();
    // Get the child's customer ID
    int tempID = tempOrder.getCustomerID();
    /* See if we have a valid CustomerID.
     * 0 indicates a new customer. */
    if (tempID == 0)
    {
        MessageBox.show("Cannot display previous "
            + "orders for a new customer.",
            this.msAppName,
            MessageBox.ICONEXCLAMATION);
        mnuPrevOrders.setChecked(false);
        return;
    }
    ordDlg.setOwner(this);
    /* Display the Previous Orders window
     * for this customer */
    ordDlg.UpdateDisplay(String.valueOf(tempID));
}
else
{
    // Uncheck the menu item
    mnuPrevOrders.setChecked(false);
    ordDlg.hide();
}
else
{
    // Get a reference to the ActiveMDIChild
    Order tempOrder = (Order) getActiveMDIChild();
    // Get the child's customer ID
    int tempID = tempOrder.getCustomerID();
    /* See whether we have a valid CustomerID
     * 0 indicates a new customer. */
    if (tempID == 0)
    {
        MessageBox.show("Cannot display previous orders"

```

```

        + "for a new customer.",
        this.msAppName,
        MessageBox.ICONEXCLAMATION);
    mnuPrevOrders.setChecked(false);
    return;
}
ordDlg.setOwner(this);
// Display the Previous Orders window for this customer
ordDlg.UpdateDisplay(String.valueOf(tempID));

```

MDIMain类定义了两个用作 Previous Orders 窗口模式的私有成员常量: POSHOWHIDE 和 POUUPDATE。调用程序将一种模式作为该方法中使用的一个参数传递给 ManagePreviousOrdersWindow 方法,来确定 Previous Orders 窗口所需的管理类型。该代码从查看模式是否为 POSHOWHIDE(指出是显示还是隐藏 Previous Orders 窗口)开始。

这部分代码首先检查是否至少有一个子窗体,并且没有选中 mnuPrevOrders 菜单项目的条件。如果满足,那么代码将显示当前活动子窗体的 Previous Orders 窗口。否则代码将隐藏该窗口,并取消选中 mnuPrevOrders 菜单项目。要显示 Previous Orders 窗口,代码首先在 mnuPrevOrders 上设置一个复选标记,获得当前子窗体的引用,然后获得客户当前的 CustomerID。如果是新客户(CustomerID 为 0),那么将出现一个消息,告诉用户新客户以前没有订单。

如果存在有效的 CustomerID 值,那么代码将 MDI 窗体变成该类 PreviousOrders 窗体私有实例的拥有者。使 MDIMain 成为 PreviousOrders 对象的拥有者不会打开 Previous Orders 窗口和 MDI 子窗口,而是使 Previous Orders 窗口成为一个工具窗口。接下来代码调用 Previous Orders 窗口的 UpdateDisplay 方法。该方法传递活动订单当前的 CustomerID。该方法(我们将在详细介绍 Previous Orders 工具窗口时讨论)处理当前选定客户订单信息的显示任务。

处理模式 POUUPDATE 的 ManagePreviousOrders 代码与模式 POSHOWHIDE 的代码相同,但是它不评价是否显示子窗体或工具窗口。代码得到活动子窗体,获得它的 CustomerID 值,并将传递给 Previous Orders 对话框的 UpdateDisplay 方法,以更新该窗口的显示。

在 MDIMain 中我们要强调的最后一个方法是 DisplayReports。回顾前面讨论的情况,该应用程序需要显示一些项目的报表,例如客户以前的订单和数据库中存储的总订单。为了在应用程序中完成这项任务,而不使用专门的报表生成器,我们使用了 Microsoft Access 来制作报表,并使用 Access 对象库的 COM 封装程序来将它们实例化。

DisplayReports 方法是 mnuCustomerReport 和 mnuAllOrders 菜单项目的 click 事件处理程序。该事件处理程序管理 Access 数据库中两个报表的显示：一个用来显示已经输入到数据库中的所有订单；另一个用来检索数据库中整个客户列表。其代码如下所示：



```

/* * Handles displaying overall reports
 */
private void DisplayReports(Object source, Event e)
{
    // Obtain a reference to the menu that was clicked
    MenuItem tempItem = (MenuItem)source;

    // Create an instance of Microsoft Access
    msacc8.Application access = new msacc8.Application();
    // Make it visible so we can see the reports
    access.setVisible(true);
    /* Open the book orders database.
       It contains the reports and queries to execute. */
    access.OpenCurrentDatabase(
        SystemInformation.getSpecialFolderPath(
            SpecialFolder.APPLICATIONDATA)
        + "\\DevWorkshop\\BookOrderManager\\BookOrders.mdb",
        false);
    if (tempItem == mnuCustomerReport)
        access.getDoCmd( )
            .OpenReport(new Variant("Customers"),
                        2,
                        new Variant(""),
                        new Variant(""));
    else
        access.getDoCmd( )
            .OpenReport(new Variant("All Orders"),
                        2,
                        new Variant(""),
                        new Variant(""));

    // Display messagebox to allow user to quit Microsoft Access
    MessageBox.show("Click OK to close Microsoft Access...",
                    msAppName,
                    MessageBox.ICONINFORMATION);
    // Close Microsoft Access
    access.Quit(0);
}

```

该方法的代码首先确定单击的是 `mnuCustomerReport` 还是 `mnuAllOrders` 菜单。它通过创建触发事件的菜单项目的实例,然后查看它等于这两个菜单项目中的哪一个来做到这一点。该代码还创建 `Access Application` 对象的实例,并将它设置为可见,以使用户能够看到它和报表。确定报表类型之后,代码将调用 `Access Application` 对象的 `OpenDatabase` 方法。该调用将通知 `Access` 我们正在使用的数据库。我们将它传递给 `Book Order Manager` 应用程序使用的 `BookOrders.mdb` 文件。

`Access` 连接到数据库后, `DisplayReports` 中的代码将使用 `Application` 对象的 `DoCmd` 对象的 `OpenReport` 方法来打开适当的报表。`OpenReport` 调用的第一个参数指定要使用的报表。第二个参数指定报表的显示方式,我们这里只是预览窗口。因为还没有在这些报表中筛选记录,所以我们使最后两个参数清空 `Variant` 对象。我们将筛选 `Previous Orders` 窗口和 `Order Details` 对话框的详细的报表中的记录。关于这些报表的详细信息将在稍后介绍。

显示报表后,将出现一个消息框,告诉用户单击 `OK` 按钮来关闭 `Access`。用户单击 `OK` 后,将调用 `Application` 对象的 `Quit` 方法来关闭 `Access`。但是强烈建议不要在使用更重要的代码之前将这样的代码添加到应用程序中。您可以通过添加引擎运行时发出的鸣叫声以示关闭。

现在我们已经探讨了 `Order` 和 `MDI` 窗体的大部分代码,下面来看一下应用程序剩余的对话框和窗口。

14.4.6 应用程序的登录对话框

虽然您的 `MDI` 窗体通常是应用程序的主窗体,但是有时希望在显示 `MDI` 窗体之前显示一个对话框来执行一些类型的操作或执行注册和登录权限的验证。因为我们的应用程序只供销售经理和销售员使用,所以它应该有一个应用程序登录功能。该功能显示一个允许用户从一个有效雇员列表中选择来进行登录的对话框。密码只有 4 位, `NumText` 控件被用来将数据条目掩饰为数字。虽然示例雇员的密码只有 4 位,但是 `NumText` 控件没有将密码字段限制为 4 个字符。否则如果黑客知道每个密码的位数,那么他们很容易就能闯入应用程序。图 14.9 显示了该对话框中运行时的情况。

因为在 `MDIMain` 窗体之前显示该对话框,并且它需要能够终止应用程序,所以它被定义为应用程序的启动窗体。以下是该对话框代码的主要部分 `CheckPassword` 事件处理程序:

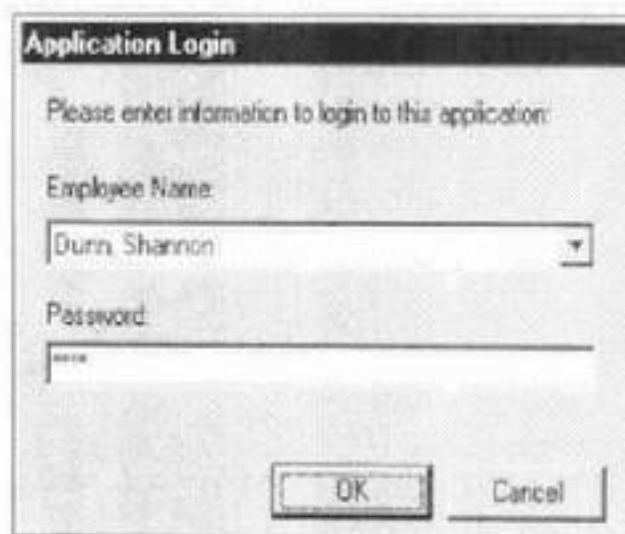


图 14.9 运行时的 Application Login 对话框



```

/* * Checks the password against the database's
   */
private void CheckPassword(Object source, Event e)
{
    // Determine whether the user has selected an employee
    if (cmbEmployees.getSelectedIndex() == -1)
    {
        MessageBox.Show("You must specify an employee name "
            + "to login.",
            MDIMain.msAppName,
            MessageBox.ICONEXCLAMATION);
        setActiveControl(cmbEmployees);
        return;
    }
    // Move to the first record in the recordset
    dsMain.getRecordset().moveFirst();
    // Find the record that the user selected in the ComboBox
    dsMain.getRecordset()
        .find("EmployeeID = "
            + String.valueOf(
                mEmpNum[cmbEmployees.getSelectedIndex()]),
            0);

    /* Determine if the selected record's password
       * matches the one the user entered */
    if (dsMain.getRecordset().getField("Password").getInt()
        == ntxtPassword.getValue())
    {

```

```

/* Load an instance of the MDI application
 * with the registry key to use */
MDIMain MDI = new MDIMain(
    Registry.CURRENTUSER
        .createSubKey(
            "Software\\"
            + "VJDevWorkshop\\"
            + "BookOrderManager"),
    (String) cmbEmployees.getSelectedItem(),
    miEmpNum[cmbEmployees.getSelectedIndex()]);
//Show the MDI Window
MDI.show();
//We are done with this window so dispose of it
dispose();
}
else
{
    /* Show messagebox to alert user
     * that the password did not match */
    MessageBox.show("Invalid password for the selected"
        + "employee."
        + "Please re-enter password.",
        MDIMain.msaAppName,
        MessageBox.ICONEXCLAMATION);
    // Set focus back to the NumText control
    setActiveControl(ntxtPassword);
    // Select all text in the control
    ntxtPassword.selectAll();
}
}

```

当加载 ApplicationLogin 对话框时,代码将 Employees 表中的所有姓名加载到用于选择雇员的 ComboBox 中。CheckPassword 的代码首先查看用户是否从该 ComboBox 中选定有效的雇员。如果没有,那么用户将得到通知,ComboBox 控件将成为活动控件。否则,代码将把对话框 DataSource 控件记录集中的当前记录移动到 cmbEmployee ComboBox 控件中选定的雇员记录。代码通过从加载对话框时创建的阵列中获得所选雇员号来移动当前记录。

CheckPassword 代码将活动记录移动到选定的雇员后,它将检查确保 NumText 控件有一个与当前所选雇员在 Employees 表中匹配的密码。如果密码匹配,那么代码将创建一个 MDIMain 的实例,并将其显示出来。在创建 MDIMain 时,代码将把指定雇员的名称和号码传递给 MDIMain 的构造函数。CheckPassword 方法最后处理 ApplicationLogin 窗体的实例。如果密码不匹配,那么该方法将通知用户,选

择 NumText 控件作为活动控件,然后退出。如果用户单击窗体中的 Cancel 按钮,那么代码将假设他忘了密码或不是有效的雇员,因此应用程序将被终止。

14.4.7 Previous Orders 窗口

正如前面讨论 Order 窗体和主 MDI 窗体时所提到的,应用程序有一个允许您查看当前所选 Order 窗体中所有以前客户订单的工具窗口。该工具窗口包含一个显示与当前客户所有以前订单相关的 ListView 控件。该窗体还包含一个用来与 Orders 表进行交互的 DataSource 控件和两个 Button 控件。图 14.10 显示了开发期间的 Previous Orders 窗口。



图 14.10 设计时的 Previous Orders 窗口

第一个 Button 控件 btnViewDetails 显示包含有关 ListView 控件所选订单指定信息的 Order Details 对话框。我们将在下一节中讨论这个对话框。另一个 Button 控件 btnViewReport 显示与我们在 MDIMain 代码中显示的 All Order 报表类似的 Access 报表。不同之处在于将显示 Customer Orders 报表,并且准则被用来将显示的记录筛选为与当前客户的 CustomerID 值匹配的订单。筛选准则将在 OpenReport 调用的最后一个参数中定义。Customer Orders 报表使用存在于 Microsoft Access 数据库文件中的内置查询。

对话框中的大量工作都是在 UpdateDisplay 方法中进行的。该方法的代码如下所示:



```
/* * Updates the display of records
 * based on the customer ID assigned
 */
public void UpdateDisplay(String customerID)
{
}
```

```

// Clear the contents of the ListView control
lsvOrders.removeAllItems( );
// Store the customerID
mcCustomerID = customerID;
/* String SQL = "Select * FROM Orders
 * WHERE CustomerID = " + customerID; */
String SQL = "SELECT Orders.OrderID, "
    + "Employees.FirstName, "
    + "Employees.LastName, "
    + "Orders.OrderDate, "
    + "Orders.ShipFirstName, "
    + "Orders.ShipLastName, "
    + "Orders.ShipAddress, "
    + "Orders.ShipCity, "
    + "Orders.ShipState, "
    + "Orders.ShipPhone, "
    + "Orders.ShipDate "
    + "FROM Customers INNER JOIN "
    + "(Employees INNER JOIN Orders "
    + "ON Employees.EmployeeID = "
    + "Orders.EmployeeID) "
    + "ON Customers.CustomerID = "
    + "Orders.CustomerID "
    + "WHERE Orders.CustomerID = "
    + customerID;
/* Assign query to data source control.
 * If records are found grid should update. */
dsOrders.setCommandText(SQL);
// Check for records
if (dsOrders.getRecordset( ).getRecordCount( ) > 0)
    UpdateListView( );
// Show the window non-modally
this.show( );
}

```

该方法通过 MDIMain 窗体得到调用, 这将传递当前所选 Order 窗体客户的当前 CustomerID 值。代码首先清除所有以前项目的 ListView 控件并存储作为参数传递的 CustomerID 值。接下来代码将创建一个内部联接的 SQL 查询, 将 Orders、Customers 和 Employee 表中的字段组合到一个包含所有要显示的信息的记录集中。

然后该查询被分配给窗体的 DataSource 控件。这里将进行一个检查, 以查看是否找到指定客户的订单记录。如果有, 那么代码将调用它的帮助程序函数 UpdateListView, 它将实际加载包含从查询检索到的 ListView 控件。最后代码显示窗体。

14.4.8 Order Details 对话框

Order 信息很有用,当客户对以前的订单有所抱怨时,他们想通知经理:除了显示指定客户以前的订单之外,还应该在 Order Details 对话框中显示订单的详细信息。图 14.11 显示了该对话框在开发期间的情况。

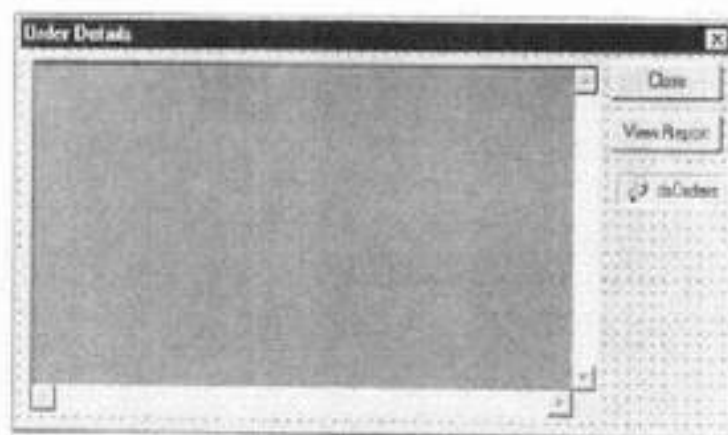


图 14.11 开发期间的 Order Details 对话框

Order Details 对话框使用包含动态列的 DataGrid 控件来显示诸如订购的书名称、单价、订购数量及总金额等信息。这些列是动态的,以允许 DataGrid 控件根据相关 DataSource 控件中的数据来创建一定大小的列。该对话框有两个按钮:用于关闭窗体的 btnClose 和像 PreviousOrders 那样显示 Access 创建的报表的 btnViewReport。

btnViewReport Button 控件的 click 事件处理程序 ViewDetailsReport 的代码与 Previous Orders 窗口中 ViewOrdersReport click 事件处理程序的代码基本相同。首先,ViewDetailsReport 中的代码在 OpenReport 调用中打开的报表是 Order Details 报表。第二,在 OpenReport 调用中筛选参数的准则在 OrderID 字段中进行筛选。当显示 Order Details 对话框时,它将从 Previous Orders 窗口接收筛选依据的 OrderID。最后,代替使用 Orders 表,ViewDetailsReport 事件处理程序中的代码使用 OrderDetails 表。

Previous Orders 窗口通过调用 OrderDetails 窗体的 DisplayDetails 方法来启动 Order Details 对话框。该方法存储 PreviousOrders 传递的 OrderID,指定 DataSource 控件中使用的内部联接 SQL 查询,以及显示窗体。与 Previous Orders 窗口不同,Order Details 对话框的记录列表由 DataGrid 控件及其与 DataSource 控件的联系管理。

14.4.9 Password Change 和 About 对话框

我们已经介绍了应用程序几乎所有的窗体和对话框,除了 Password Change 和 About 对话框。图 14.12 显示了运行时的 Password Change 对话框,图 14.13 显示了运行时的 About 对话框。由于它们的代码很少,所以仅解释一下这些对话框的作用。

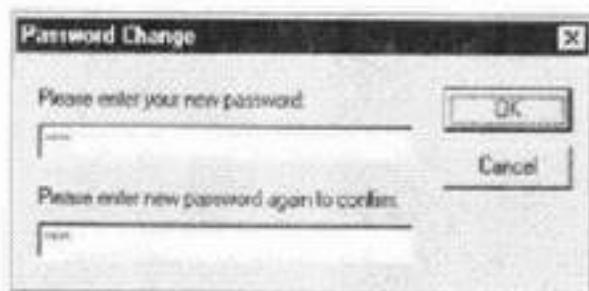


图 14.12 运行时的 Password Change 对话框

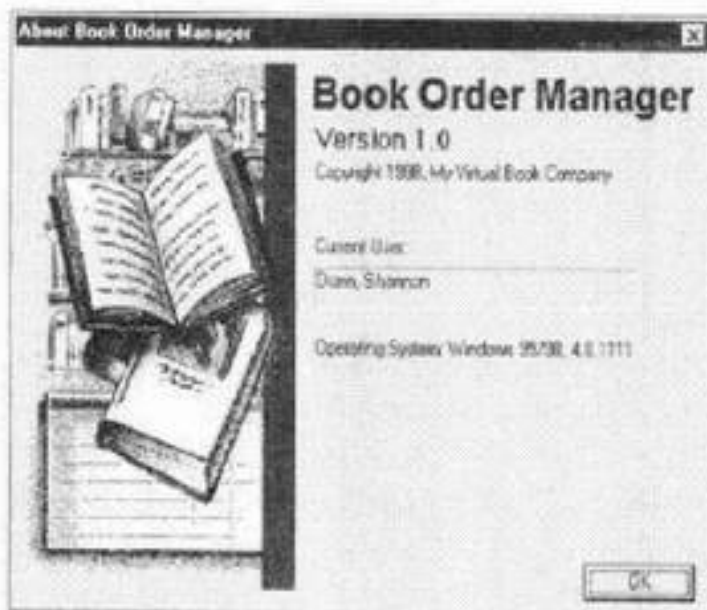


图 14.13 运行时的 About 对话框

Password Change 对话框通过 MDIMain 中 mnuPassword 菜单项目的 click 事件处理程序来调用。该对话框显示两个 NumText 控件,一个允许您指定新的密码,另一个用来确认所做的更改。当在对话框中单击 OK 时,将会比较密码。如果它们匹配,那么在数据库中对当前登录的用户进行更改。因为用户必须登录才能操作应用程序,所以不需要请求旧的密码。

图 14.13 所示的 About 对话框是一个标准的应用程序 About 框,并且当用户单击 MDIMain 中的 mnuAbout 菜单项目时显示它。该对话框非常酷,因为它显示当前登录的用户名称,以及正在运行该应用程序的操作系统版本。如果您试图跟踪错误并需要了解正在运行应用程序的 Windows 版本,那么操作系统版本将非常有用。

现在我们已经完成了开发过程及运行应用程序所需的代码。在创建类似的应用程序时,您应当考虑在订单开发列表中添加每个窗体和元素(除非涉及的内容有所不同)。在下一节中,我们将逐步介绍我们的应用程序,以使我们能够涉及更多的功能,并将与讨论的代码融会贯通。

14.5 程序运行

现在我们来运行应用程序,以便您可以更好地理解我们刚刚讨论的代码,并体会一下程序的所有功能。在 Visual J++ 中编译并运行应用程序。如果有什么问题,那么要确保所有的 COM 组件都在“注册表”中得到注册。(如果需要,可以使用 Regsvr32.exe 对所有需要注册的 COM DLL 进行这样的操作。)当显示应用程序时,将看到 Application Login 窗体。从列表中选择名字(Dunn, Shannon),然后输入“1234”作为密码。

这时对话框将消失,然后出现显示有雇员名称的应用程序启动屏幕。请注意,在启动屏幕中可以称这个应用程序为 Order Manager。这个称呼非常适合该应用程序。

关闭启动屏幕后,将会看到 MDIMain。除了 New Order 按钮之外,大多数的工具栏按钮都被禁用。单击 New Order 按钮,显示 CustomerSearchEngine2 COM 组件的对话框。选择 Select Last Name 作为搜索字段,然后输入“Dunn”作为要搜索的姓。列表中有许多 Dunns。在列表中找到名字,然后单击 OK 按钮。在 Customer Information 选项卡中将显示一个包含自己数据的新 Order 窗体。

单击 Order Information 选项卡,然后单击 Add 按钮,将一个项目添加到订单中。这时将出现 New Order 对话框。单击它的 Browse 按钮,显示 BookSearchEngine COM 组件的对话框。选择 Title 作为搜索字段,然后输入“Visual”作为要搜索的图书。您将在搜索结果中看到这本书。选定它,然后单击 OK。

回到 New Order 对话框,控件将用有关的订单信息(如单价、数量和总金额等)填写。将数量更改为 14,然后注意在数量方面的每一个变化,Total Price 将体现这种变化。设置完数量,单击 OK。关于订单项目的信息将被添加到 ListView 控件中。再次单击 Add,以同样的过程搜索以“Art”开头的书。您将会看到相关

的图。将这本书的一个副本添加到订单中,然后单击 OK。

当返回到 Order 窗体时,您将会注意到有两个总计项目。第一个项目显示一堆书,第二个项目只显示一本书,因为我们在该订单项目中只订购了一本书。还要注意总计在 ListView 控件下的显示方式。

现在单击 Order 对话框的 Shipping & Payments 选项卡。将会得到提示,将客户信息移动到发货信息中。继续下面的操作。将发货日期设置为 4/4/2000,将 Ship Method 设置为 Mail Company,并将 Freight 设置为 3.00(不要输入美元符号)。注意,Total Amount Due 字段现在包含订购项目的总金额及运费。选择 Armenian Express 作为 Payment Type,然后将一个杜撰的信用卡号输入到 Credit Card Number 字段中。在 Expiration Date 字段中输入“14/14/89”。单击 OK,保存记录。您将会得到通知,信用卡已经过期。将这个值更改为未来的一个日期,但还不要单击 OK。

单击 View 菜单中的 Previous Orders,显示 Previous Orders 工具窗口,并且包含所有以前的订单。从列表中任意选择一个项目,然后单击 View Details 按钮,显示包含订单特定信息的 Order Details 对话框。在 Order Details 对话框中,单击 View Report 按钮。这时将显示 Microsoft Access,并且显示 Order Details 的报表。单击 OK,关闭 Microsoft Access,然后单击 OK,退出 Order Details 对话框。

现在打开另一个新的订单,再次在 CustomerSearchEngine2 对话框中搜索 Dunn——但这次选择 Jonathan Dunn,然后单击 OK。当出现新订单窗体时,注意 Previous Orders 窗口现在显示 Jonathan 的信息。在两个订单之间进行切换,注意 Previous Orders 对话框每次的变化情况。单击 OK,保存订单。

MDIMain 底部的状态栏将通知您,订单被保存,并且窗口被关闭。关闭订单窗口可以防止保存它后影响订单。请记住,公司要求您只允许经理能够更改订单。通过在保存订单后删除 Order 窗口,可以满足这个要求。

现在已经介绍了所有的功能。您可以运行一下应用程序来看一看 Password Change 对话框及 All Order 和 All Customer 报表。现在可以打包这个程序,并将它提交给公司以得到审批。

如果应用程序需要得到一些专业人士的审批,那么建议您通过使用 Project Properties 对话框 Output Format 选项卡中的 self-extracting Setup 打包选项,将它及支持文件打包到一个正式的安装程序中。您可以单击对话框中的 Advanced 按钮,显示安装程序的选项。确保在您的安装程序中包括 Microsoft Virtual Machine for Java,以使安装程序中包括 VM 和 WFC 库。使用正式的安装程序可以省去很多麻烦。在将应用程序发送给每个人之前,还要测试一下整个安装过程。

您的应用程序正准备得到公司的审批,但很可能需要在开发后对其进行一些更改来完成它。以下或许是您需要注意的一些要求,以使应用程序能够更好地

地适应销售竞争。

- ◆ 因为该应用程序最终要用于 SQL Server, 所以需要将这此报表移到更为正式的报表工具。My Virtual Book Company 是正在研究的报表工具, 所以您应该考虑使用这个工具。作为选择, 您还可以使用 Access 数据库, 并简单地附加到 SQL Server 中来创建报表。这种解决方案比较经济, 但并不推荐使用它, 因为在 Access 中通过附加到 SQL 服务器来生成报表中只是一种应急方法。
- ◆ 为应用程序提供不同级别的输入内容。公司也许希望提供一些只有具有一定特权的人才能访问的功能。可以更新 Employee 表来适应这些变化。
- ◆ 提供一种不在订单形式下搜索图书和客户的方法。这可能需要创建只显示搜索对话框的菜单和事件处理程序。
- ◆ 保持登录到应用程序中的用户记录。如果出现安全性问题, 这可能是非常重要的。
- ◆ 提供一种存储 Customers 中使用的最新的支付方式, 如信用卡。这可以加速订单创建的过程。
- ◆ 创建一个自定义控件, 允许您筛选 Shipping & Payments 选项卡中的 Freight 字段, 但只允许能够转换为双精度数据类型的值。



更上一层楼

在本章中, 学习了如何将我们讨论的各种功能组合成一个相当大的实际应用。您可以为一家虚拟的公司创建一个订单条目程序, 该应用程序必须满足特定的要求, 了解到在编写代码之前如何为项目创建一个规划, 您还可以创建一个开发顺序列表, 其中显示要实现的对话框和功能, 及其实现时间。

我们以与添加窗体和功能相同的顺序介绍了应用程序的代码。您还了解到如何运行时操作程序并查看其功能。最后, 您学习了一些关于发布应用程序的技巧, 并探讨了一些扩展程序及添加公司要求的其他功能的方法。